A STRUCTURED AUDIO SYSTEM BASED ON JAVAOL

Yi-Song Siao, Alvin W.Y. Su, Jia-Lin Yah and Jian-Lung Wu

Studio of the Computer Research on the Music & Media (SCREAM)
National Cheng Kung University, Tainan, Taiwan, R.O.C.

ABSTRACT

Structure Audio (SA) is an algorithmic based coding technology designed for low bit-rate high quality audio. With this technology, the desired sound can be identical on both the encoder side and the decoder side. However, the SAOL defined in the MPEG-4 standard is not as powerful as Java language. Therefore, in this paper, we present a new way of implementing synthesis algorithms, named JavaOL to replace the original SAOL language. In this paper, we present a client-server SA system based on JavaOL, too.

1. INTRODUCTION

Structured Audio contains several parts for different functions described as follows. The first part is the synthesis algorithms for sound generation and processing. The second part is the score data. MPEG defines a SA standard called MPEG-4 Structured Audio (MPEG-4 SA) [1]. The synthesis language defined in MPEG-4 is called the Structured Audio Orchestra Language (SAOL) [2]. SAOL is a C like language, but it has lots of special syntax to obtain a little bit of convenience. Therefore, it's difficult to implement the SAOL compiler. Based on our previous experience with Java, it is currently one of the suitable ways to fulfill the promise made by SA because Java is general enough to implement almost any synthesis algorithms and close to SAOL, too. In this paper, we discuss the JavaOL and propose such a Java based decoder. To demonstrate the power of SA, we also propose a SA framework. This framework is an open architecture and anyone can write decoders on it. The current JavaOL decoder presented in this paper is such an example. With the JavaOL decoder, the system performance is greatly improved. However, Java is still not as efficient as the applications implemented by C language. Therefore, we provide another decoder: a Wavetable decoder [3][4]. It's implemented with C++ and ASM for better performance.
The rest of the paper is organized as follows. The brief introduction of the SA system is given in section 2. In section 3, we discuss how the Java is applied on media processing. In section 4, the concepts and roles of JavaOL are described. In section 5, we talk about the proposed system and the performance comparison with sfront and SAOLC is provided. Future works are in section 6.

2. SA ARCHITECTURE

Since the limitation of the bandwidth of physical networks, the data size of multimedia data is always restricted if the real-time requirement has to be met. The ways commonly seen for decreasing data size are compressing and/or describing the sound data as follows.

2.1.Compression-Type Technologies

MP3[5][6] and MPEG-2/4 Advanced Audio Coding (AAC) [7] are among the most popular techniques. The compression ratio of 11:1 or even higher can be achieved though sound quality is compromised with higher compression rate.

2.2.Description-type Technologies

In MIDI [8], data such as pitch, volume, and other description information are stored. One disadvantage is that the discrepancies among different MIDI synthesis devices exist. To eliminate the difference among synthesis devices is much desired.

2.3.The SA Approach

The Machine Listening Group [9] proposed a description-type approach called Structured Audio. In SA, not only note and control information, but also the sound synthesis algorithms are described. Each note in the score needs to be calculated into raw PCM data through the algorithm specified to it (Fig 1). The streams of raw data from all tracks are summed and the result is passed into the wave output device.

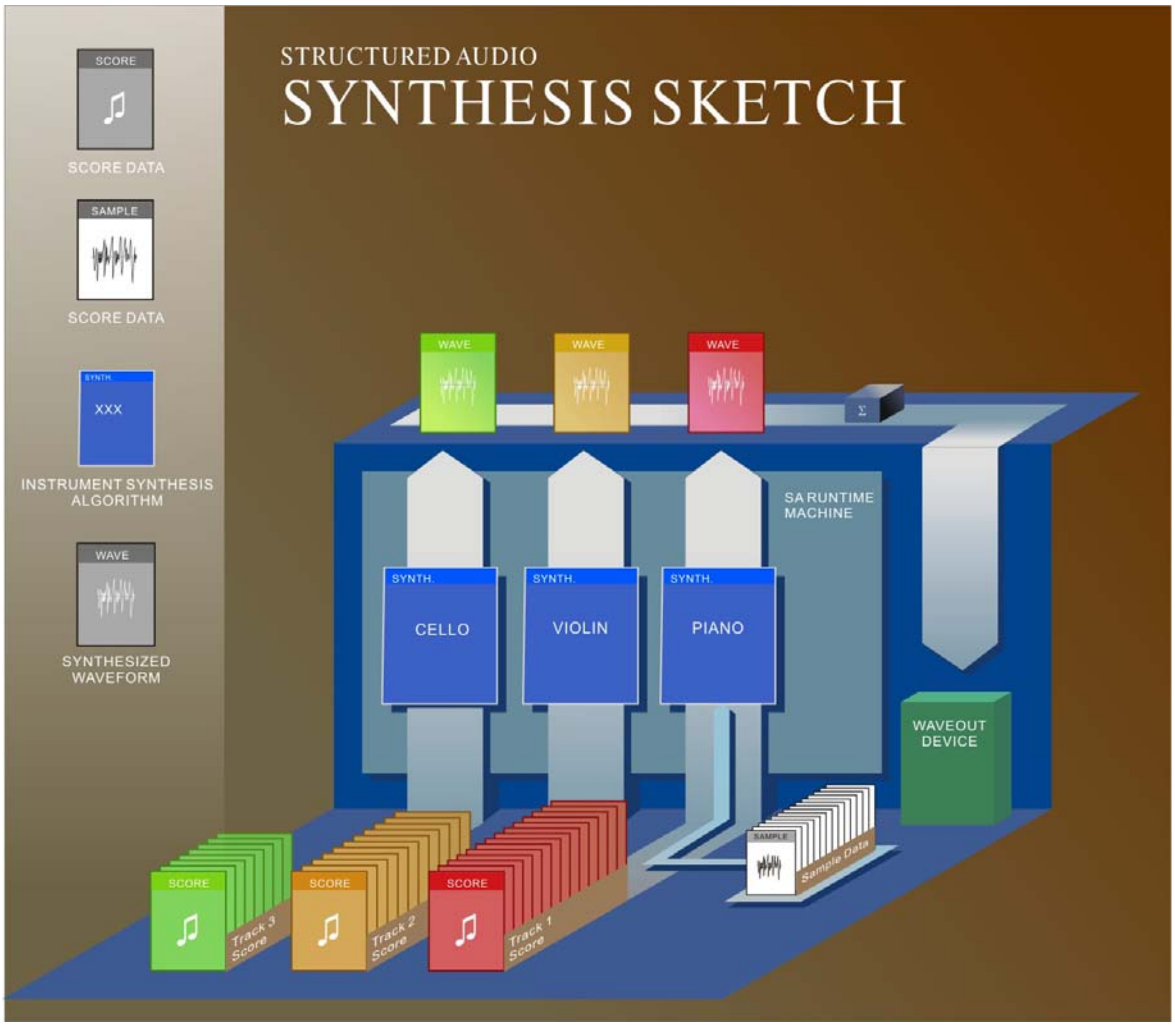


Fig 1.A SA Synthesis Example.
There are 3 tracks containing score data, and 3 synthesis algorithms in a data stream. The score data is *translated* into raw PCM data with the corresponding synthesis algorithm.

3. JAVA BASED MACHINE FOR MEDIA PROCESSING

JavaOL is a Java based media processing language. Basics, merits, demerits and some possible refinements for Java based media processing are discussed in the following context.

3.1.Basics of Java

Java code is compiled to intermediate binary code rather than a machine dependent code. When the code is to be executed, it is loaded by Java Virtual Machine (JVM) [10]. Java Platform contains two parts: JVM and Java Application Programming Interface (Java API). Java API is a well-designed library includes many useful and optimized packages, such as math, I/O, GUI, Network etc… A Java programmer can use these packages to save much programming time.

3.2.Java to Apply Media Processing

Quite a few media processing libraries are developed for Java. For example, audio codec, and video libraries are well developed by many companies and institutions. *Java Media Framework* (JMF) [11]-[13] is one well-known library developed and maintained by Sun and IBM Haifa.

3.3.Advantages and Disadvantages

Java is free and has many free tools available on the internet. Second, new ideas and bug-fix come out more quickly. Third, Java is platform independent and portable. Fourth, the Java code can be compressed as Jar files, and Jar files are executable without uncompressing by user. Finally, Java code enjoys high reusability and this saves time for system development.
The major disadvantage of using Java is about its efficiency. In many cases, programmers would rather write native code (platform-dependent) than Java code for higher performance. In addition, Java codes need Java VM to execute. Though there are many operation systems providing Java VM, the Java VM is sometimes an optional component and portability becomes a problem. Finally, documents of Java libraries are sometimes hard to find.

3.4.Possible Improvement of Java Based Media Processing System

To improve the efficiency, microprocessor for efficient Java Computing (MAJC) is provided by Sun. Hot-Spot [14] optimization technology in the compiler will further improve run-time performance. Furthermore, MAJC can run applications compiled into its native instruction set from other languages, such as C or C++. To solve the portability problem, Microsoft has released his own JVM (Microsoft Java Virtual Machine) and Sun provides Windows version Java VM [15]-[17]. The *Java Media Framework* is a powerful library for media processing and saves much time for programmer. Win32, Solaris, AIX and Linux versions are now available though the Macintosh version is still absent.

4. J AVA AS AN ORCHESTRA LANGUAGE

4.1.Java as a Signal Processing Language

The quintessence of SA is that the synthesis methods are contained in the transmitted data. Therefore describing the synthesis methods becomes an important issue when implementing SA. Because JAVA has many virtues, we choose Java as the solution in this project. Implementation of instrument synthesis algorithms in Java is possible by following a set of rules defined in this paper (JavaOL), and compile these Java codes into Java binary files (usually the so-called "class" file). Therefore, these binary files can be transmitted to the client sides for real-time synthesis and playback.

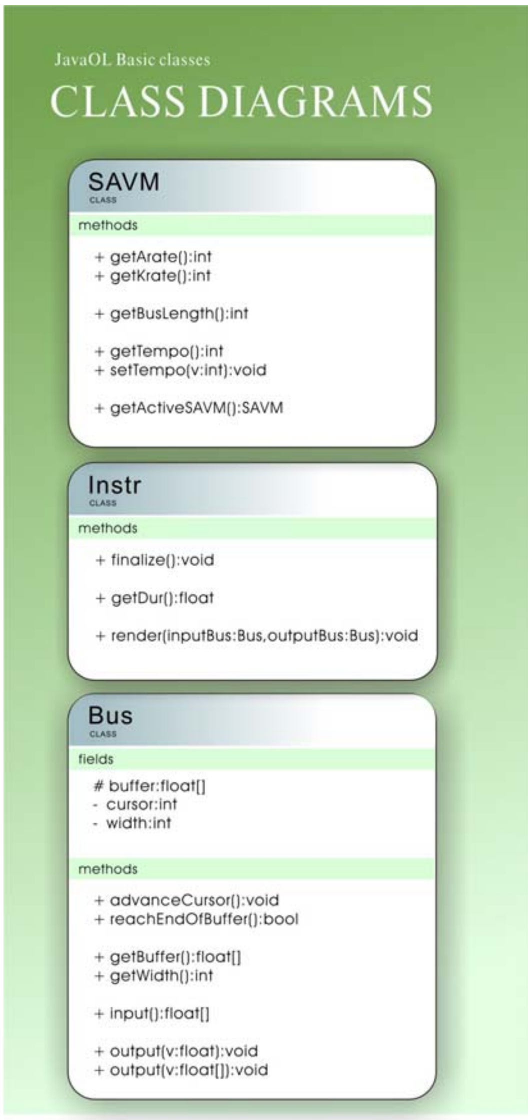


Fig.2 Class Diagrams of 3 JavaOL basic classes.

4.2.JavaOL opcode

JavaOL is a set of classes for implementing the synthesis algorithms. Fig.2 shows the UML class diagrams of 3 basic classes defined in JavaOL.

class SAVM

The **SAVM** (Structured Audio Virtual Machine) class is an interface to access the system information. The SA system creates one **SAVM** object before the synthesis process begins, and releases it when the process is done. The public methods are listed as follows:

getArate	Retrieves the system a-rate.

getKrate	Retrieves the system k-rate.
getBusLength	Retrieves the length of system bus. (We will discuss the concept of bus latter.)
getTempo	Retrieves the system tempo in beats per minutes.
setTempo	Sets the tempo of system.
getActiveSAVM	Retrieves the active **SAVM** object.

Here is an example to get the system a-rate.

```
int systemArate=getActiveSAVM().getArate();
```

class Instr

The **Instr** class is the base class for all JavaOL instrument classes. An **Instr** object is created in a k-cycle of SA system. To initialize the object specific variables, one needs to write codes in the class constructor method. The public methods are listed as follows:

getDur	Retrieves the duration in beat.
render	When the Instr object is active, the SA system calls the **Instr** render method to fill the output buffer. A user defined instrument synthesis class should override this virtual function to fill the output buffer.
finalize	Called by the SA system when the **Instr** object is to be released .

Here is an example of implementing a Piano instrument class:

```
class Piano extends Instr
{
        public Piano(int p1,float p2)
        {
                // initialize Instr variables here
        }
        public void render(Bus inputBus, Bus outputBus)
        {
                // fill the outputBus here
        }
}
```

class Bus

The **Bus** class is used for data transmission between an **Instr** object and the SA system. It contains a 32-bits floating number array in order to store audio signal. When the SA system calls the render method of **Instr** objects, it passes two parameters (input bus and the output bus) into the render method. In Fig 3, we illustrate the concept of these buses. Fig 3(A) is the concept of accessing a buffer of a 2 dimensional array. The bus width corresponds to the number of channels, and the bus length is a constant for a session. To calculate the bus length, the following formula is used: bus_length= a-rate/k-rate. Instrument programmers can obtain this value by calling the “getBusLength” method of **SAVM**. Fig 3(B) shows the allocation of data in memory. All channel data is arranged in an interleaving manner. The render method of an Instr object has to fill the whole buffer during each call.

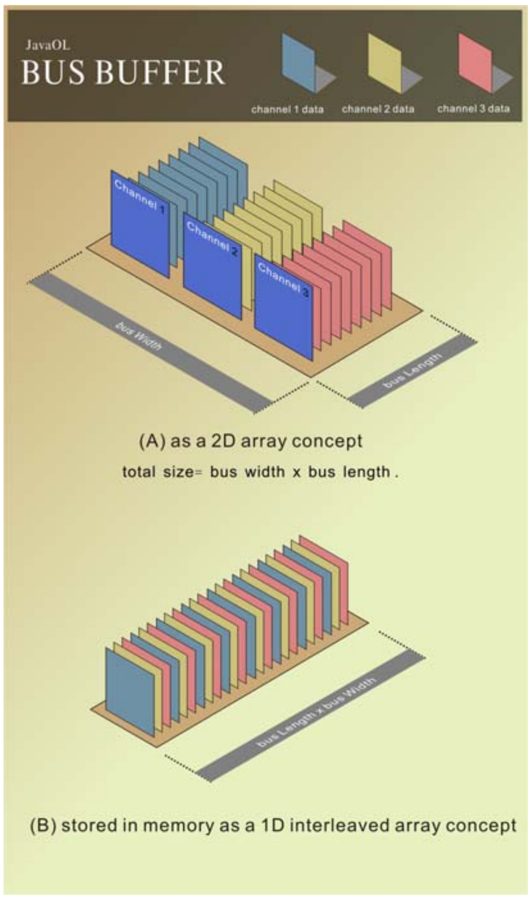


Fig 3. Bus buffer concept in JavaOL.

The fields are listed as follows:

buffer	A float point number array to store synthesis data.
cursor	Store the read/write position.
width	Store the width of the bus.

The methods are listed as follows:

advanceCursor	Increase the read/write position.

reachEndOfBuffer	Retrieves a Boolean value indicates whether the read/write cursor reaches the end of buffer.
getBuffer	Retrieves the buffer field. Usually the user should not directly access the buffer array by getting it with this method; the only reason is for efficiency.
getWidth	Retrieves the bus width (channels).
input	Read data from the buffer. The return value is a float array.
output	(2 overrides) Output data to the buffer.

Here is an example of filling a bus buffer for data of 3 channels:

```
class Piano extends Instr
{
        public void render(Bus inputBus, Bus outputBus)
        {
                // fill the outputBus
                while(!reachEndOfBuffer)
                {
                        float v[3];

                        //calculate the value of v
                                :
                                :
                                :

                        //output data to the buffer
                        output(v);
                        //advance the cursor
                        advanceCursor();
                }
        }
}
```

5. IMPLEMENTATION OF JAVAOL

The programs implemented in the proposed system include 3 main components: Player, Server and Score Editor. This system works under the client-server architecture. The SSA-Player in a client site links to the SSA-Server online and receives streamed data to calculate the sounds for further playing back the music. We will discuss these 3 elements and the data transmitting protocol in the following context.

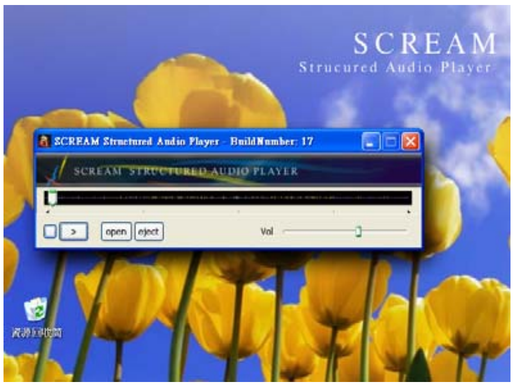

SCREAM
Strucured Audio Player
SCREAM Structured Audio Player - BuildNumber: 17
SCREAM STRUCTURED AUDIO PLAYER
open
eject
Vol

Fig 4. The SSA-Player screenshot.

5.1.The SSAPlayer

The "SSA-Player" is the player for SSASBS (Scream Structured Audio Bitstream). When a user clicks the SSASBS link, this program starts automatically and opens the SSASBS link file to get the connection information. Then the player opens a connection to the specific SSA-Server, sends request , gets the streamed data, and plays the song.

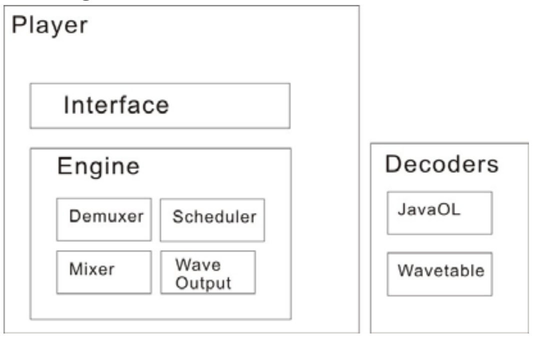
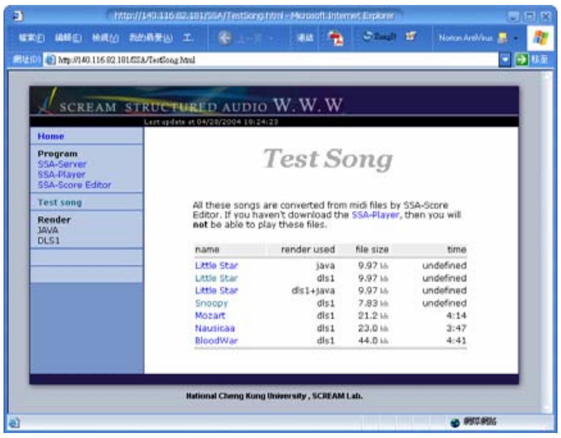


Fig 5 The SSA-Player architecture.

The architecture of the SSA-Player is shown in Fig 5. The player is presented with a friendly user interface shown in Fig 4 as well as the internal synthesis engine. The internal synthesis engine is the core of this SA system. It is also called the "Scream SA Engine" or SSAE. We'll discuss it below.


http://140.116.82.101/SSA/TestSong.html - Microsoft Internet Explorer
SCREAM STRUCTURED AUDIO W.W.W
Last update at 04/28/2004 10:24:23
Home
Program
SSA-Server
SSA-Player
SSA-Score Editor
Test song
Render
JAVA
DLS1

Test Song

All these songs are converted from midi files by SSA-Score Editor. If you haven't download the SSA-Player, then you will **not** be able to play these files.

name	render used	file size	time
Little Star	java	9.97 kb	undefined
Little Star	dls1	9.97 kb	undefined
Little Star	dls1+java	9.97 kb	undefined
Snoopy	dls1	7.83 kb	undefined
Mozart	dls1	21.2 kb	4:14
Nausicaa	dls1	23.0 kb	3:47
BloodWar	dls1	44.0 kb	4:41

National Cheng Kung University , SCREAM Lab.

Fig 6 Screenshot of the test song page of the project.[http://140.116.82.181/SSA/TestSong.html]

Each decoder is packed into a DLL file and it should export some specific functions to communicate with the proposed SSAE. We currently provide Wavetable and JavaOL decoders in the project. In the SSA test page (Fig 6), we provide 7 test songs. The decoders used by each song are listed at the second column.

5.2.The Scream SA Engine (SSAE)

The SSAE includes the following parts:

Demuxer

Since every track event is sliced according to the unit time, different track events of the same time slice are packed altogether. If we are going to play the song, we need to de-multiplex the packs and put the events into the right tracks.

Scheduler

Scheduler is a time based note processing system. Its job is to pick the events from the tracks according to the execution time tags marked for the events.

Mixer

A note is rendered into PCM raw data in floating point format. To combine the sound of notes at the same slice, a Mixer is required.

Wave-output

The Mixer exports the data through the Wave-output to a sound card.

5.3.Interactions between SA-Engine and Decoders

In the SSASBS bitstream, there is a table indicates which decoder should be used. The SA-Engine loads these decoders from the respective DLL files and sends initialization messages to them. At each k-pass cycle, the engine renders each note in the active event buffer by sending render messages to the respective decoders. Upon receiving the render message, the specific decoder renders the note into raw data for one time slice. When the SSASBS bitstream is fully decoded, the decoders will receive finalization messages and release the allocated resources.

5.4.The JavaOL Decoder

The JavaOL decoder is like a bridge between the SSAE and the JVM (see Fig 7). All Java instructions are executed on the JVM. When the SSAE needs to execute score commands, these commands are sent and handled by the JavaOL decoder, and the JavaOL decoder has to determine which Java Instructions is to be executed. For example, if the score indicates that a piano note should be played at time T_0, the scheduler of the SSAE will generate a command to create a piano note instance at T_0. The JavaOL decoder receives the command and then executes the "create instance" method of the Java VM to create a piano note instance.

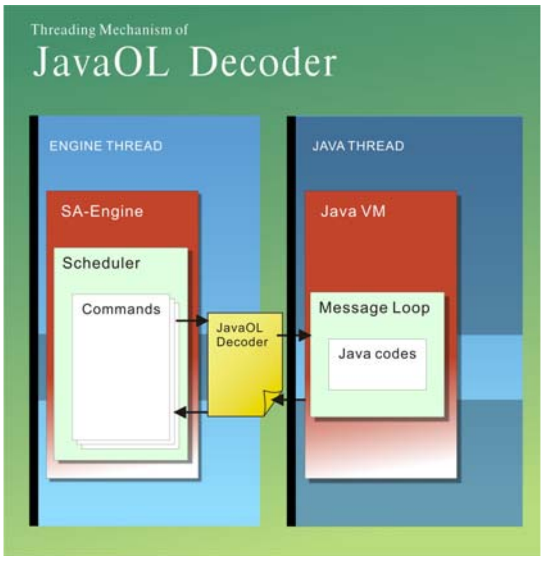


Fig 7 The threading mechanism of JavaOL decoder

5.5.The Wavetable Decoder

The default Wavetable decoder supports the DLS1 [18] file format. Some test songs in the SSA test page (Fig 6) use the Wavetable/DLS1 decoder.

5.6.Client-Server Architecture and Protocol

The SSAS reported in this paper uses the TCP to transmit data rather than UDP. The reasons to use TCP are: the data size is small compared to today's internet bandwidth; the data used to synthesize the musical sound shouldn't be lost. Otherwise, it is difficult to recover or even conceal the error. When a client connects to an available port on the server, it creates a session for the user. The communication protocol between the server and clients is relatively easy. Each message sent from clients has a message ID field. The ID is used to identify which command should be executed upon receiving this message.

5.7.Performance Analysis

Currently, the most popular tools have been the standard SAOLC [19] and sfront [20]. In order to compare the performance among these implementations, we use a simple algorithm to test their performances. The Karplus-Strong plucked string model is used in this paper [21]. The SASL part and the SAOL part are shown in Appendix 1 and Appendix 2, respectively. Both SAOLC and sfront are used for comparison. The third program is generated from converting the SAOL program to a JAVA program automatically using the proposed tools. This program is shown in Appendix 3. The fourth 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. For all the programs, srate is set as 44100 and krate is set as 100. The duration is 4 seconds. These programs are executed on a Pentium 4 2.0G/1024MB personal computer running Window XP. The execution times of the four cases are shown in Fig 8.

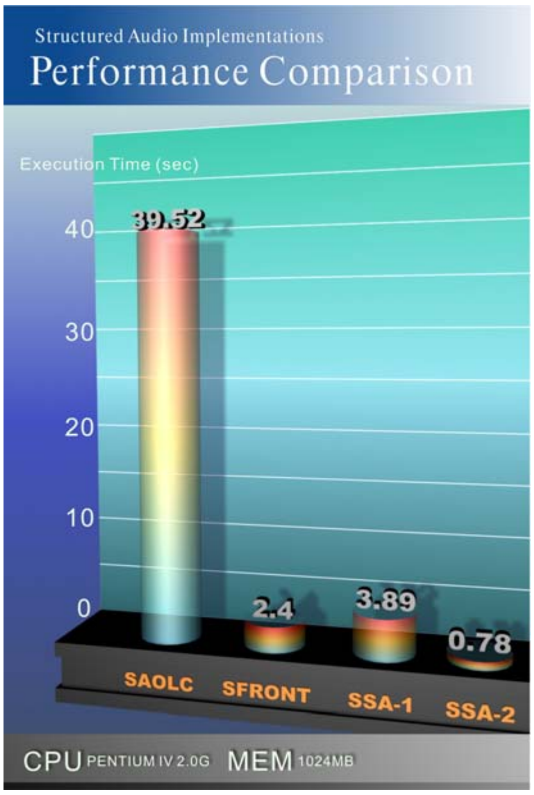


Fig 8 Performance comparison

6. CONCLUSION AND FUTURE WORKS

A SA based client server system, SSAS, with **JavaOL** is presented. Without the disadvantages of conventional MPEG-4 SAOL, real time internet SA based audio applications becomes possible with the proposed SSAS. Because SSAS is implemented with Java, it can be widely supported by many computer systems. Through its open framework, users can write their own synthesis programs as well as their own decoders which can work flawlessly with our SSAS if they follow some **JavaOL** rules. The proposed system does fulfill the promise made by the SA. Currently, the system software can be downloaded at [22]. The documentation about SSAS will be improved shortly. However, to make SSAS more powerful and convenient to use, the following works are expected. They are discussed as follows.

6.1.SSA-Server

In the future, we should provide the real-time broadcasting support in the server. Score information (SASL) recorded from live performance will be imported and packed into data packages at SSA-Server.

6.2.Accommodation of various audio formats

To accommodate more data formats as MP3 and put data in various formats into a single data stream, one needs to define a data format. Furthermore, we need to prepare corresponding decoders for sound data of different audio formats in the SSA-Player.

6.3.The SSASBS protocol

The network data transmission should use the UDP to achieve the real-time broadcasting purpose in a more efficient way. By specifying which datagram is important, servers can discard unimportant datagram when the network is congetsed.

6.4.Porting JavaOL Decoder into other Media Platforms

Though the proposed system is currently only available under Microsoft Window environment, it is not operated under Microsoft Media player. It is our goal to allow it to function under multimedia players of different operating systems so that **JavaOL** can be more popular.

7. APPENDIXES

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
{
   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;
		}
}
```

8. REFERENCE

[1] Eric D. Scheirer, "The MPEG-4 Structured Audio standard," in Proc. IEEE International Conference on Acoustics,Speech and Signal Processing (ICASSP-98), 1998

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

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

[4] 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.

[5] Mp3' Tech - Mp3Pro/SBR,
[http://www.mp3-tech.org/sbr.html]

[6] Robert F. Easley, John G. Michel, Sarv Devaraj, The MP3 open standard and the music industry's response to Internet piracy.

[7] MPEG AAC (Advanced Audio Coding)
[http://www.iis.fraunhofer.de/amm/download/mpeg_aac.pdf]

[8] The MIDI MANUFACTURES ASSOCIATION
[http://www.midi.org]

[9] MIT Media Lab Machine Listening Group
[http://bop.media.mit.edu/]

[10] T. Lindholm and F. Yellin. The Java Virtual Ma-chine Specification. Addison-Wesley, 1997

[11] Java Media Framework
[http://java.sun.com/products/java-media/jmf/index.html]

[12] IBM HotMedia
[http://www.software.ibm.com/net.media/]

[13] IBM MediaBeans
[http://www.software.ibm.com/net.media/solutions/mediabeans/]

[14] White Paper-The Java HotSpot Performance Engine Architecture
[http://java.sun.com/products/hotspot/whitepaper.html]

[15] Microsoft Java Virtual Machine downloads
[http://www.itnet.org/ms/msjavx86.exe]

[16] Microsoft Java Virtual Machine technical information
[http://www.microsoft.com/mscorp/java/]

[17] Microsoft SDK for Java v4.0 downloads
[http://www.bumpersoft.com/Programming/Java/D_978_index.htm]

[18] DLS level 1 Specification from www.midi.org
[http://www.midi.org/about-midi/dls/dlsspec.shtml]

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

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

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

[22] Scream Structured Audio Player
[http://140.116.82.181/SSA/SSAPlayer.html]