Application of high quality internet music service & combination of MPEG-4 Structured Audio and physical model synthesis method

Adviser: Associate Professor Alvin W.Y. Su
System Developer: Jian-Lung Wu, Yi-Song Xiao
Developing Tools: Visual C++, Borland C++ Builder
Linux KDevelop C++, Visual J++
Operating System: Windows XP, Redhat Linux 7.3

1 Project Purpose

1.1 Problem Definition:

Both of the low bandwidth and the large size of music files are the major problem of listening to the Internet music. Even though more and more people use ADSL service, the low bandwidth is common at the Internet rush hour. Although MP3 files are compressed files, they are still not small enough. Is there any music file format that is not affected by low bandwidth and is small enough to be downloaded quickly?

1.2 Our Solution:

We want to build a web site that provides many classical music files by using Streaming technique. Therefore, users could listen to the music as the player receives the data stream. Moreover, these music files should small enough. Although the MP3 file could be transmitted by Streaming technique, its size is too big. The MIDI file's size is small, but it cannot be transmitted by Streaming technique.

Hence, we choose MPEG-4 Structure Audio (SA) as our music files format. Basically, SA contains two parts: ***1) SAOL (Structured Audio Orchestra Language)***; ***2) SASL (Structured Audio Score Language)***. The SAOL 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 algorithm. 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.

Normally, the player should use SAOL files to generate sound samples, but we have no synthesis algorithms of 128 instruments. (Any one of the algorithms could be another project.) Therefore, we use the wave-table of the sound card to generate sound samples.

2 Program Analysis

2.1 Website and Database Construction:

We use PHP + MySQL to construct the website and database. First, we buy the classical music MIDI files from foreign website. Then, these MIDI files are converted to SASL files and built in the music database. In the major table, we save full data on the non-duplicate column (For example, the file's name column), and save ID on the duplicate column (For example, the author's name column). There is relation table recording the corresponding author the ID represents. This method could reduce the search time, because number comparison is faster than words comparison. For example, users want to search all music composed by Bach. The system finds the Bach's ID is "3". Then, it searches "3" on the author's column in major table.

Because the amounts of music files are too large (more than 6000 classical music files), we write a batch file to get the file information from original website and write it into database automatically.

In our website, we just put a file link instead of a real music bit stream file. After receiving the file link, the player uses this file link to request the server program developed in Linux to send the real bit stream file. Why not put the bit stream file in the website? The main reason is that we get the file control only after IE browser downloads the "full" file. It violates the definition of streaming.

2.2 Convert MIDI to SASL:

2.2.1 Parse MIDI:

We extract the event list of the MIDI files according to the MIDI file format as following steps.

I. Open the MIDI file.
II. Read the 4-byte ***type***, **"MThd"** (type of header chunk), the 4-byte ***length***(length in bytes of the chunk data part), 2-byte ***format***(MIDI file format), 2-byte ***track***(number of track chunks), 2-byte ***division***(defines the default unit of delta-time for this MIDI file)
III. Read 4-byte ***type***, **"MTrk"** (type of track chunk), 4-byte ***length***((length in bytes of the chunk data part)
IV. Read all MIDI event data according to the ***length***, and put them into array.
V. Go back to step **III,** until reading all tracks' data.
VI. Decode the MIDI event data of all tracks, and put them into appropriate channels.
VII. Finish

2.2.2 MIDI to SASL:

To convert the MIDI to SASL, we combine major MIDI events, ***Note On***、***Note Off***、***Program Change***, to form SASL event, ***Instr***. Other MIDI events, ***Polyphonic Key Pressure*** 、***Controller Change*** 、***Channel Key Pressure***、***Pitch Bend***, are combined to form SASL event, ***Control***. Here is the converting table.

Midi Event	SASL Event	Comment
Note On	*Instr*	Trigger time of *Instr* is decided by *Note On*, and duration of *Instr* is decided by *Note Off*. *Instr's* parameter 1、 parameter 2 are the key and velocity separately.
Note Off		
Polyphonic Key Pressure	*Control*	In SAOL, there should be a corresponding parameter that controls the attribute of polyphonic key pressure. Otherwise, we do not need to convert this MIDI event.
Controller Change	*Control*	The same as *Polyphonic Key Pressure*
Program Change	*Instr*	*Instr's* name is changed by *Program Change*
Channel Key Pressure	*Control*	The same as *Polyphonic Key Pressure*
Pitch Bend	*Control*	The same as *Polyphonic Key Pressure*
	End	*End* is decided by the music length

※SOSL has no attribute of "Channel", and all *Instr* events are independent. In order to maintain the attribute of "Channel", we add the string, "//TrackLine", to represent the separation of the "Channel".

2.3 Encode SASL to Bit Stream:

2.3.1 Parse SASL:

SASL has five kinds of events: *Instr, Control, Tempo, Table, End*.

Event	Syntax	Comment
Instr	*[label :] trigger name dur p1 p2…*	*Instr* event creates a new instance of a SAOL instrument. *"label"* is optional. The SASL *Control* event uses *label* to reference the instance. *"trigger"* is the starting time for the instance. *"name"* is the name of the SAOL instrument to instantiate. *"dur"* is the duration of the instance. A value of -1 indicates indefinite duration. *"p1, p2…"* are the instrument parameters.
Control	*trigger [label]* ***control*** *var number*	*"trigger"* is the time to execute the *Control* event. *"label"* is optional text label that links the *Control* event to an *Instr* event. *"**control**"* is a keyword. *"var"* is the name of the SAOL variable to change. *"number"* is the new value for the variable.
Tempo	*trigger* ***tempo*** *number*	*"trigger"* is the time to execute the *Tempo* event. *"**tempo**"* is a keyword. *"number"* is the new value for the tempo.
Table	*trigger* ***table*** *var gen p1 p2…*	*"trigger"* is the time to execute the *Table* event. *"**table**"* is a keyword. *"var"* is the name of the SAOL table to change. *"gen"* is the wave table generator to use to initialize the table. *"gen"* may also be the keyword *"destroy"*, which destroys any existing wave table created for this wave table name. *"p1, p2…"* are the parameters for the wave table generator.
End	*trigger* ***end***	*"trigger"* is the time to end the SAOL simulation. *"**end**"* is a keyword.

Only the SASL files converted by us have the Channel separation string "//TrackLine." Therefore, we can put every event in its corresponding channel. Otherwise, we put all events in one channel. (It is OK, because SASL originally has no attribute of Channel.) In order to reduce the amount of data transmission, the names of variables of ***name***、***label***、***var***、***gen*** are all assigned a specified ID and these relations are saved in individual tables. (These tables should be transmitted first.) In client, we look the ID up in these tables to find the real name of variables.

2.3.2 SASL to Bit stream

In SASL files, events may not be stored in time sequence. Therefore, all events should be sorted before streaming. After sorting all events, the ***name***、***label***、***var***、***gen*** tables are added to the bit stream first. Next, the amounts of all time blocks are added. (We set one second as one time block.) Third, we add the amounts of all events in first time block and events' data to the bit stream. Then, we add next time block's data until all time blocks' data are added to the bit stream. The bit stream formats are illustrated as following.

name 、***label*** 、***var*** 、***gen*** Tables

Description	The amounts of ***name*** table's strings		The amounts of string 1 characters		Character 1	Character 2	……
Bits Length	4	1	5	1	6	6	……

This is a check bit. For example, if the amounts of name table's strings are more than that 4 bits can represent, this check bit will be set 1. Therefore, the next 4 bits are combined with the original 4 bits to represent the amounts of name table's strings.

The amounts of ***label*** table's strings		The amounts of string 1 characters		Character 1	Character 2	……
4	1	5	1	6	6	……

The amounts of ***var*** table's strings		……	The amounts of ***gen*** table's strings		……
4	1	……	4	1	……

Time Blocks

Description	The amounts of all time blocks		The amounts of all events in first time block		The events' data in first time block
Bits Length	9	1	6	1	……

The amounts of all events in second time block		The events' data in second time block	……
6	1	……	……

Instr (Type=0_2)

Description	Type	Check	***label***		***trigger***(decimal part)	***name***		Check	***dur***(integer part)	
Bits Length	1	1	7	1	9	4	1	1	3	1

If this bit is 0, it means that the ***dur***'s value is -1. We don't need to transmit the following bits of ***dur***'s part.

If this bit is 0, it means that there is no ***label*** part. We don't need to transmit the following 8 bits.

dur(decimal part)	The amounts of parameters		***p1***	***p2***	……
9	4	1	32	32	……

Control (Type=10_2)

Description	Type	***trigger***(decimal part)	Check	***label***		***var***		***value***
Bits Length	2	9	1	7	1	7	1	32

If this bit is 0, it means that there is no ***label*** part. We don't need to transmit the following 8 bits.

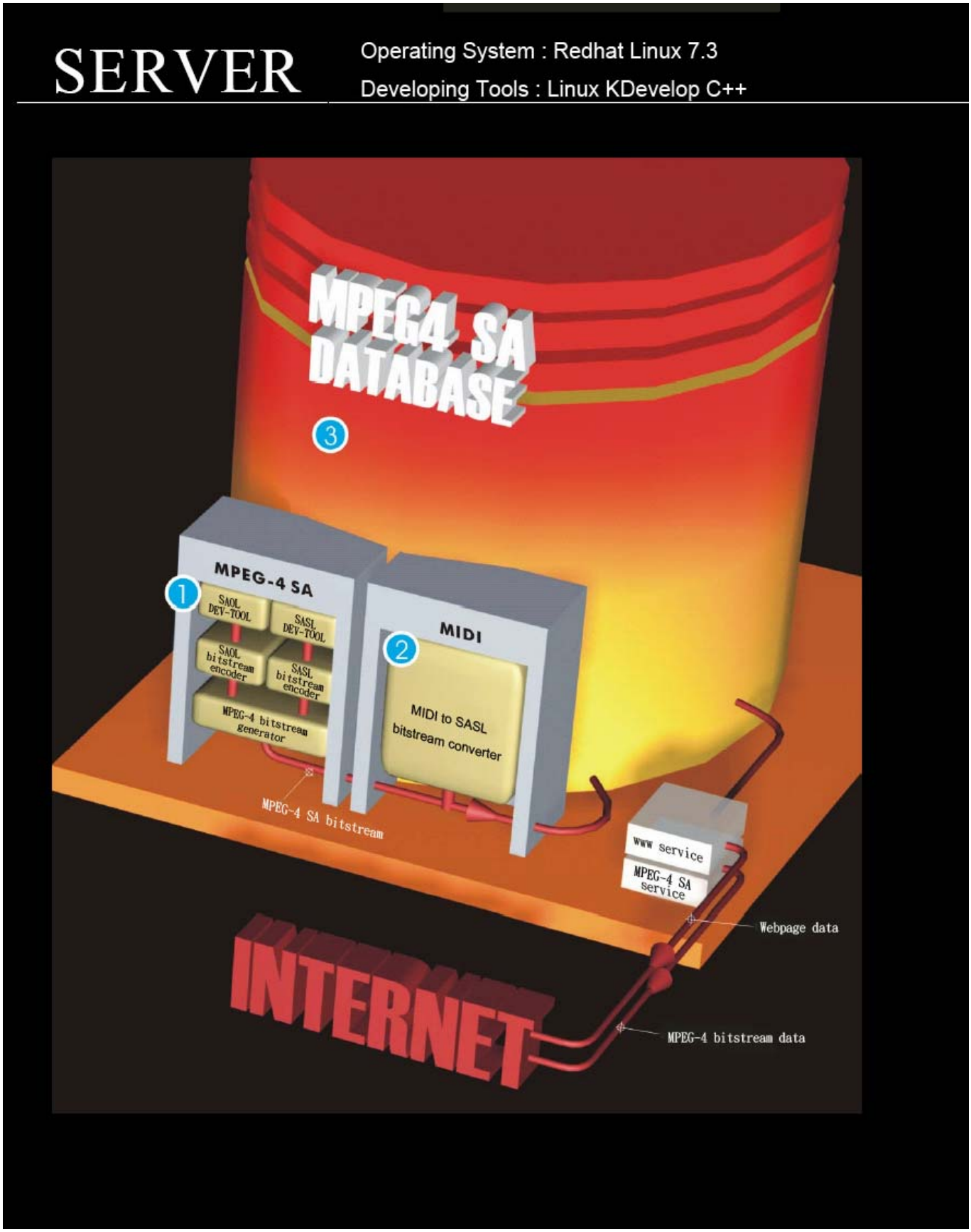
Table (Type=11_2)

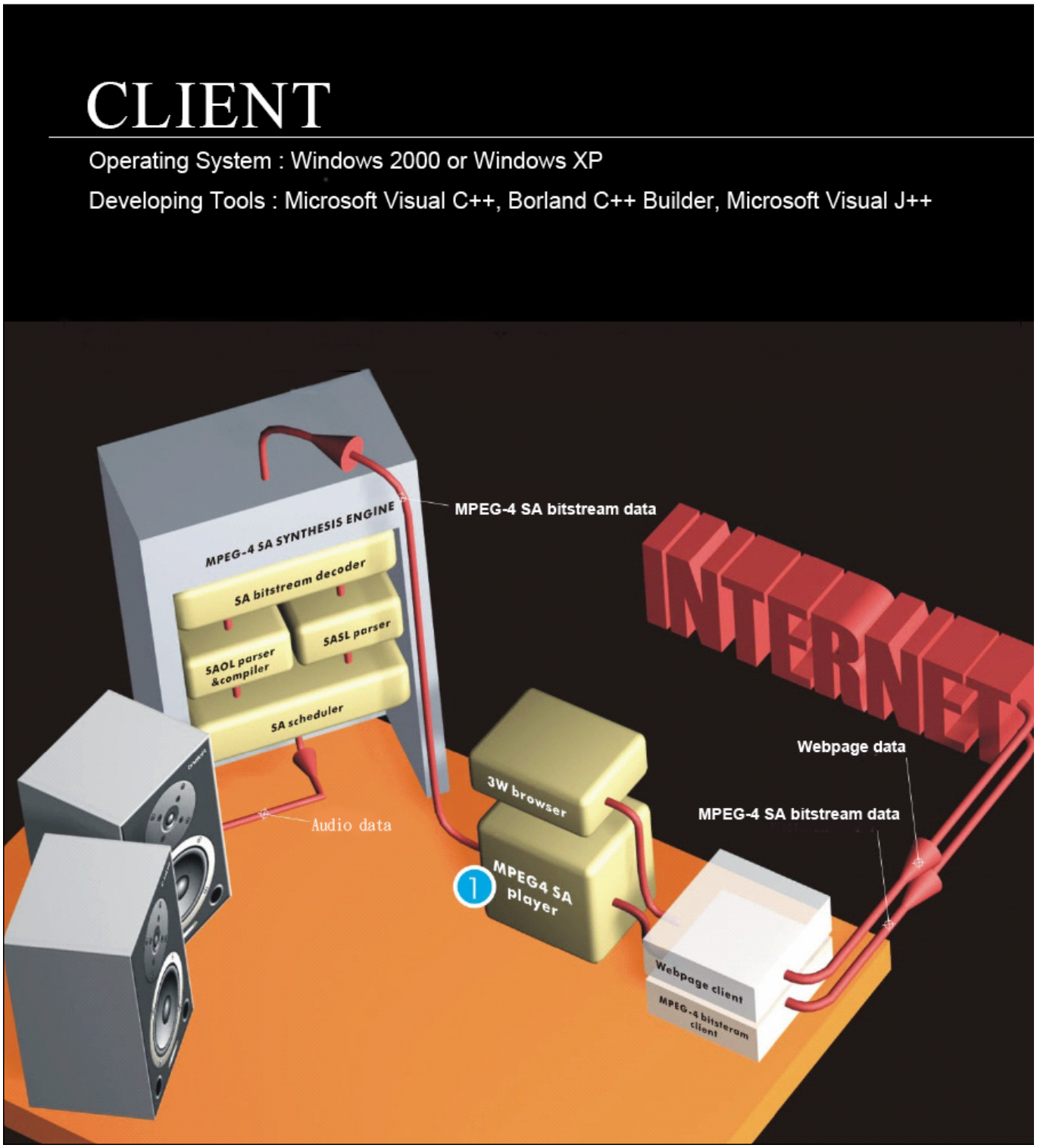
Description	Type	***trigger***(decimal part)	***var***		***gen***		The amounts of parameters		***p1***	***p2***	……
Bits Length	2	9	7	1	3	1	3	1	32	32	……

※ Because the absolute time of every event has been calculated before streaming, we do not need to transmit the ***Tempo*** events. Moreover, because the amounts of all time blocks have been transmitted first, we do not need to transmit the ***End*** events.

※ The encoding & decoding programs explained above are all saved as ***DLL (Dynamic Link Library)*** format. When the programs in client side or server side need these encoding or decoding functions, they can use these ***DLL*** files.

3 System Chart





4 Conclusion

The MPEG-4 SA greatly reduces the size of music files, and the streaming technique shortens the waiting time of listening to the Internet music. In the future, we want to join the SAOL part and the music synthesis algorithms to our system. As a result, the music quality will be much more improved, and everybody could listen to the higher-quality Internet music in lower bandwidth.