SMS Composer and *SMS Conductor*: Applications for Spectral Modeling Synthesis Composition and Performance

Eduard Resina
Audiovisual Institute, Pompeu Fabra University
Rambla 31, 08002 Barcelona, Spain
eduard@iua.upf.es

Abstract

SMS Composer and SMS Conductor are Windows applications designed to take musical advantage of both the flexibility and potential of SMS sound transformations and its current capability for real-time synthesis. SMS Composer offers a powerful compositional environment for SMS score generation, editing and synthesis. SMS Conductor is a real-time SMS controller, mainly focused on the problem of tracking and translating conductors' queues into time-stretch synthesis fluctuations.

1 Introduction

Spectral Modeling Synthesis (SMS) [1], provides a musically attractive and powerful tools for sound transformation. The increasing speed of CPUs makes it possible, and highly appealing, to apply this system to new and more complex musical situations than to what it could be attempted some time ago. Through 1998, some applications designed to control and manipulate the synthesis parameters of SMS analysis files have been developed at the Audiovisual Institute of the Pompeu Fabra University of Barcelona. SMS Composer and SMS Conductor form, together with SMS Performer (an application by A. Loscos and E. Resina presented in another paper in these proceedings) [2], a trilogy of such SMS control related applications.

Where as SMS Performer has been designed for real-time performance (real-time parametric control and synthesis), SMS Composer stands by a different and somehow opposed kind of musical premises and demands. SMS Composer is a score (list of events or parameter specifications for the synthesis of some SMS analysis file) generator, editor and synthesizer. It is not designed for real-time synthesis, but rather for detailed editing of the events' synthesis parameters.

On the other hand, SMS Conductor stands conceptually somewhat in between SMS Performer and SMS Composer. SMS Conductor is also designed to perform real-time synthesis of an SMS file, yet, most of the conditions defining the program's behavior during performance must be entered and edited in advance.

Obviously, there is no redundant work at all in these applications since each one of them addresses problems concerning completely different contextual needs and situations.

2 SMS Composer

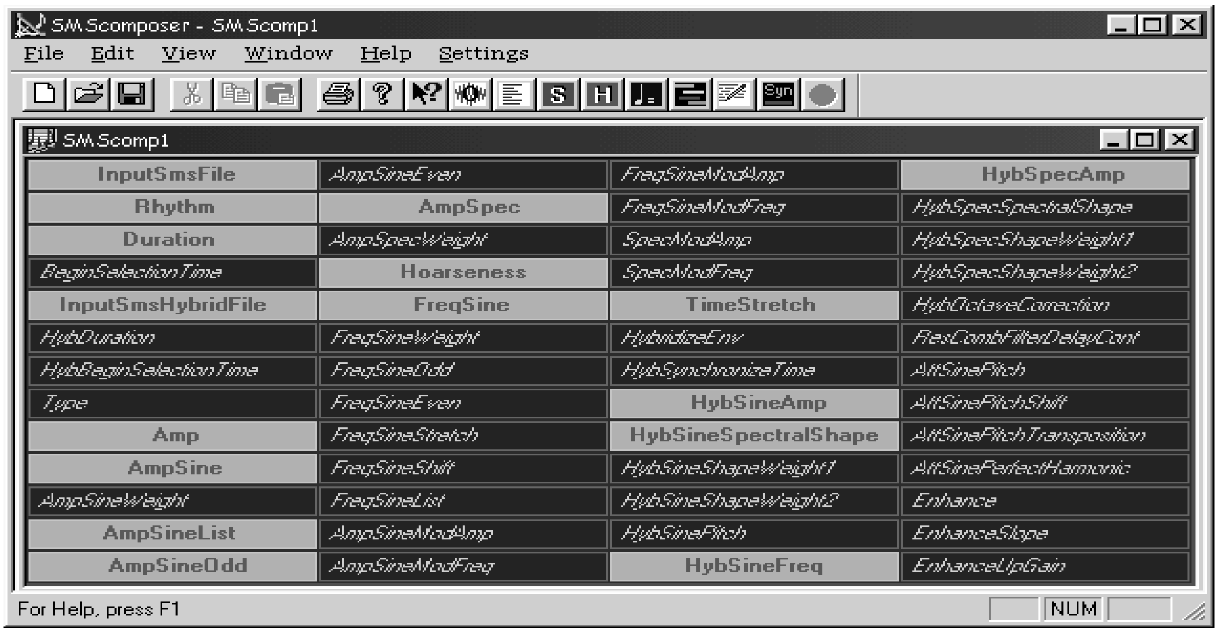
The design of SMS Composer responds both to demands derived from previous compositional experience using SMS [3][4][5], and to the influence of some other score generation environments used through these last few years (especially J. Rahn's Lisp Kernel [6] and R. Taube's Common Music [7]) for CSound score generation, as well as the Lisp embedded list philosophy which turns to fit so nicely into the proliferation of musical levels so natural to compositional thinking.

2.1. From the beginning

When the user starts an SMS Composer new session, an ordered list or map of all the available synthesis parameters appears on screen. When the application writes the score file, only the selected (highlighted) parameters will be taken into account. (*Figure 1*)

Before any of the parameters can be edited, the user will be prompted to enter the path of at least one SMS file. In order to edit any of the hybridization parameters the user must enter at least another SMS file for this purpose.

Both InputSmsFile and InputHybridFile dialog boxes can register up to twenty different SMS files. The user will be able to assign any of these synthesis and hybridization files to each individual event or group of events to be generated in the score.



InputSmsFile	AmpSineEven	FreqSineModAmp	HybSpecAmp
Rhythm	AmpSpec	FreqSineModFreq	HybSpecSpectralShape
Duration	AmpSpecWeight	SpecModAmp	HybSpecShapeWeight1
BeginSelectionTime	Hoarseness	SpecModFreq	HybSpecShapeWeight2
InputSmsHybridFile	FreqSine	TimeStretch	HybOctaveCorrection
HybDuration	FreqSineWeight	HybridizeEnv	ResCombFilterDelayConf
HybBeginSelectionTime	FreqSineOdd	HybSynchronizeTime	AttSinePitch
Type	FreqSineEven	HybSineAmp	AttSinePitchShift
Amp	FreqSineStretch	HybSineSpectralShape	AttSinePitchTransposition
AmpSine	FreqSineShift	HybSineShapeWeight1	AttSinePerfectHarmonic
AmpSineWeight	FreqSineList	HybSineShapeWeight2	Enhance
AmpSineList	AmpSineModAmp	HybSinePitch	EnhanceSlope
AmpSineOdd	AmpSineModFreq	HybSineFreq	EnhanceUpGain

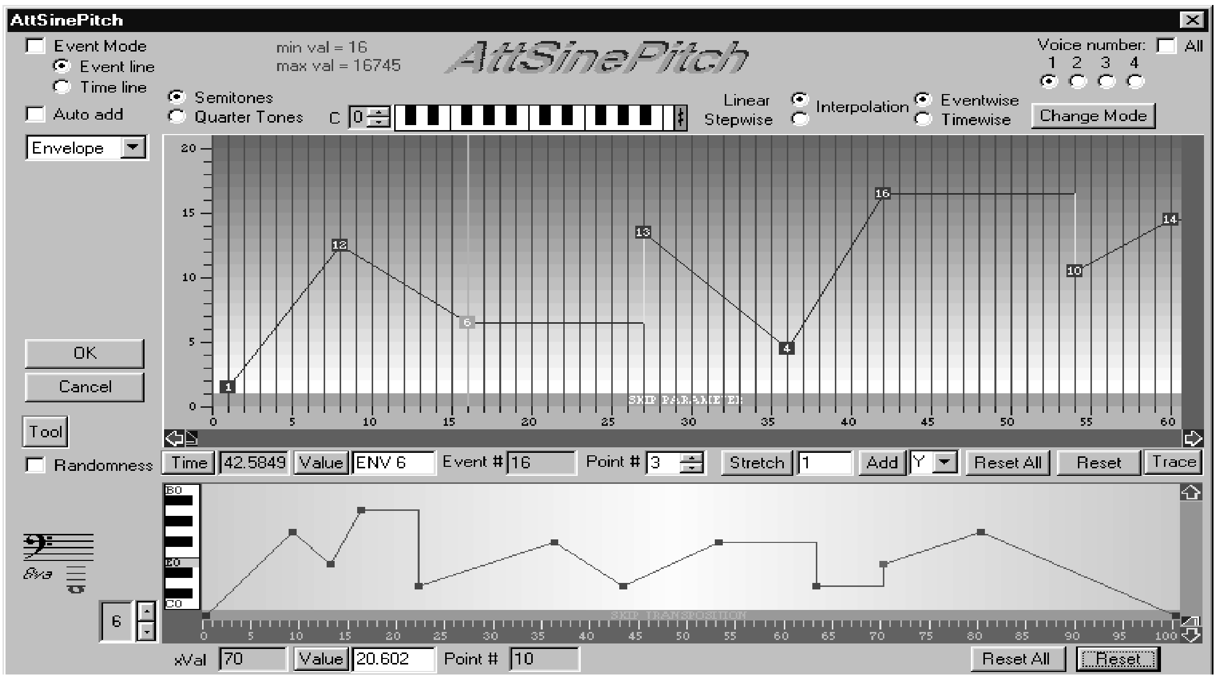
Figure 1

2.2 The Envelopes

By double-clicking on any of the parameters on screen, an envelope window with a double eventline vs. timeline) mode is shown. Most of SMS parameters can take either a single value or an envelope as input. The window for these parameters can be switched to envelope type of input.

In such a case, the points on the main envelope refer to one of the twenty secondary envelopes the user can define per parameter. Every parameter's main envelope can edit up to four independent voices or sequences of events. Every voice has an event or time offset, as well as an independent amount of randomness which will be applied to the voice's values if selected. (*Figure 2*)

The points can be entered directly into the envelope though mouse clicking, by playing on a MIDI keyboard, or (in the case of the various frequency and duration parameters) by mouse clicking on a



min val = 16
max val = 16745

Time 42.5849 Value ENV 6 Event # 16 Point # 3 Stretch 1 Add Y

xVal 70 Value 20.602 Point # 10

Figure 2

graphical keyboard representation, or by selecting musical notation duration values which can represent a large range of rhythmic combinations including triplets, quintuplets and septuplets. When using musical notation for the durations (instead of absolute decimal notation), the user will be able to change the duration ratio by modifying the global tempo at will. Events between points can either interpolate or hold the last point's value, and a number of editing tools to set and modify the value of a group of points is provided.

2.3 Lists and sublists

The recursive nature of SMS Composer is implemented in such a way that every EnvPoint object owns a pointer to a list. This is, every point can become a list, and every List object can hold any number of points which in turn can become sublists. This way the user can descend down a tree of some twenty levels of sublists.

SMS Composer inherits some of the item stream types implemented in Common Music (cycle, sequence, rotation, palindrome, random, etc.) The user can keep track of the ramifications and visualize a representation of the sublist tree by pressing the TRACE button. The graph will show the type of list, its period, as well as all its descendants. (*Figure 3*)

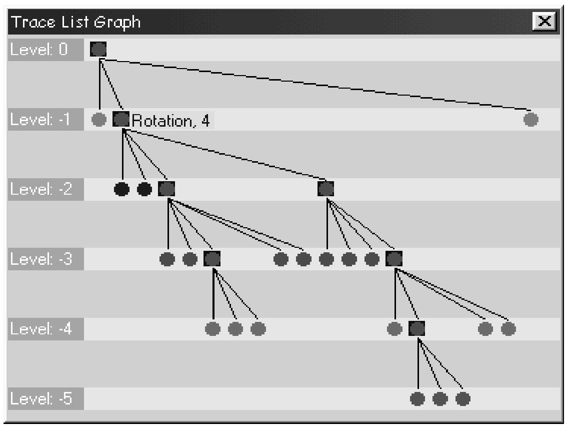


Figure 3

2.4 Ending the session

Once the score has been generated, it can be edited from within the application. The editor can search any voice's event either by event number or by event time, making editing of the score fast and easy.

Finally, the user can synthesize the score and listen to the resulting sound from within SMS Composer, or select a sound editor. In this latter case, SMS Composer will automatically open the selected sound editor and load the resulting sound file when the synthesis is done. When writing to an existing sound file the user can select to overwrite the file or to write on top of the sound data mixing the newly synthesized sound to the previously existing one. By saving the session the user has no need to keep the score after the sound has been synthesized. The score can be instantaneously recreated when opening the session, and all the settings and editing will still be there for further modifications.

A number of additions are being incorporated to the synthesis capabilities of SMS which obey to emerging compositional demands that have come out while implementing SMS Composer or while composing with SMS. One of them is the possibility of generating stereo sound files and the addition of panoramics as a new SMS synthesis parameter. Until now SMS could only generate mono sound files.

Even though SMS Composer has been designed to suite the demands of composing with SMS, the structure of the application makes it easy to adapt it to other environments in order to generate different types of scores or events. Indeed, a version of the application for generating CSound scores will be available in short.

3 SMS Conductor

SMS Conductor takes advantage of the recent possibility of running SMS in real-time. The main idea is for a conductor to be able to control the time-stretch synthesis parameter in performance at the same time the sound is being generated, without having to be especially trained, by simply marking, with clarity, the moments of inflexion where he wishes to modify the pre-established tempo.

The input to SMS Conductor is an SMS inharmonic analysis of a sound file. The analysis file is not loaded into cache memory (as does SMS Performer in order to optimize the reading into the file). Since the analysis of the sound file to be conducted will probably be large, SMS Conductor cannot take advantage of this resource. Real-time synthesis will still work in a Pentium Processor at 266 MHz even if the analysis file is not loaded into cache memory. In order to optimize speed, time stretch frame interpolation is disabled, which should not make much of a difference considering that, in general, tempo deviations are not expected to be enormous.

One of the important handicaps at this experimental estate of SMS Conductor is that SMS real-time synthesis works only on mono sound files. One way around this problem would be to have a different computer for the synthesis of each of the channels. Since minor synchronization problems could be expected, SMS Conductor establishes a synchronization protocol that allows SMS to know the exact runtime position so that it can be corrected when running out of phase.

Indeed, SMS Conductor should be running on a different computer to the one doing the synthesis, even when synthesizing one single analysis file. SMS Conductor must receive, filter and process incoming MIDI messages, the messages must then be evaluated in relation to the expected tempo at that given moment in the performance to calculate the time stretch value to be sent to SMS. Furthermore, the screen must be continuously redrawn to display the advance of the music and the updated values referring to the performance and the tempo fluctuation statistics. All this tasks would make it very hard for SMS real-time synthesis to work properly unless running on a separate computer. The ideal setting for SMS Conductor in performance is shown next. (*Figure 4*)

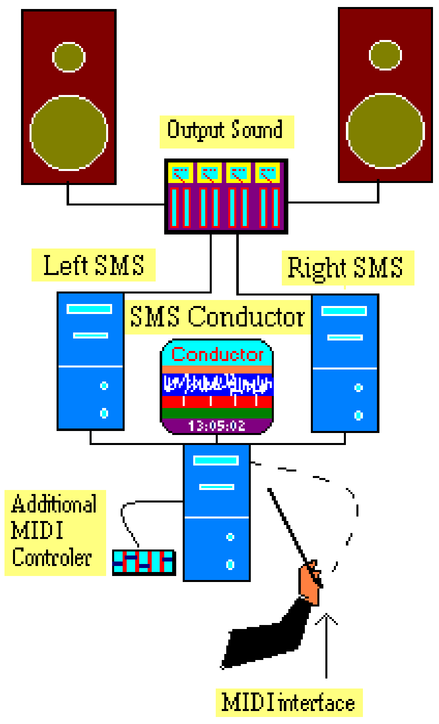


Figure 4

It is needless to say that it is simply a matter of time before a single personal computer can handle all this tasks by itself.

3.1 Input data

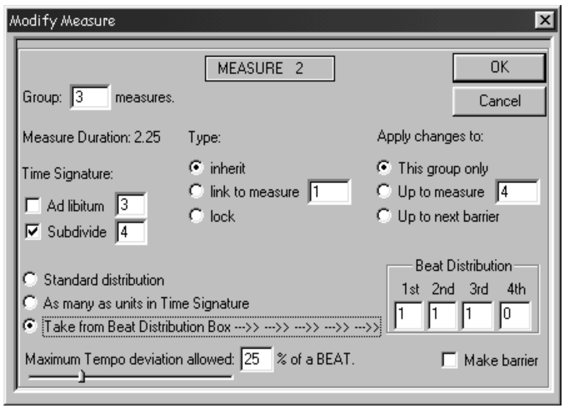
While editing, SMS Conductor can either take an SMS analysis file or a sound file for sound wave graphical display purposes. When the user saves a session and the SaveWaveData option is selected, the minimal sound data required for the application's display is saved together with the session (441 Hz, 8 bit data). This is useful because a 20 minutes mono sound file at 44.1Khz takes about 100MB hard disk space, while the corresponding SMS analysis file (without any compression) would take up to 350MB. The display wave data would take around 5MB instead, freeing the user from having to work where the SMS or sound file is stored, or forcing him to move this huge amount of data back and forth. Obviously, if the sound file is loaded, it will not be possible for the user to listen to any sound transformation. Only the original sound can be listened to. If only display data is loaded, nothing can be played, yet the user will still be able to visualize the amplitude and time stretch transformations. It is possible to work on a session even if no wave data has been loaded, but the sound wave display is very useful to check whether the music score matches specific sound events, allowing the user to edit the score data (tempo, measures, etc) in accordance.

3.2 Measure attributes

When starting a new session, the user selects an initial tempo, time signature, subdivision and measures' beat distribution, as well as the default maximum tempo deviation allowed in performance (if this latter value is set to zero, then there is not much of a reason to use SMS Conductor at all). Once the session is started, each measure's settings can be modified at will by double clicking on the square with the corresponding measure's number. An editing dialog box will be displayed. (Figure 5)

A measure can be made the head of a group of measures. In this case, any modification that affects the head measure will be applied to the whole group of measures.

Measures can be set to three different modes: *inherit* (the default), *linked*, and *locked*. Inherit mode means that when you modify any of the measures in a session, the modifications might be applied to all the measures in inherit mode following the edited one.



Modify Measure

MEASURE 2

OK

Cancel

Group: 3 measures.

Measure Duration: 2.25

Type: [✓ inherit] link to measure 1 lock

Apply changes to: [✓ This group only] Up to measure 4 Up to next barrier

Time Signature:

☐ Ad libitum 3

☑ Subdivide 4

Standard distribution

As many as units in Time Signature

[✓ Take from Beat Distribution Box --->> --->> --->> --->> --->>]

Beat Distribution			
1st	2nd	3rd	4th
1	1	1	0

Maximum Tempo deviation allowed: 25 % of a BEAT.

☐ Make barrier

Figure 5

Linked mode means that a measure (or group of measures) is linked to some previous measure (or group of measures). Modifications on the latter will be applied to the linked one. Locked mode means that a measure (or group of measures) can only be modified by directly editing it.

When editing a measure, the user can decide how far down in the inheritance chain the changes must be applied. Furthermore, a measure can be made a *barrier* to stop modifications descending through the chain. The inherit-mode measures beyond a barrier will only be affected by modifications to a measure located also after that barrier.

The user can change the time signature of a measure or group of measures, make it subdivisible, and select the number of beats per subdivision. If the measure is subdivided, subdivision marks will be displayed on the measure. In this case, SMS Conductor will pay attention to conductor's queues within the limits of a subdivision. The deviation is then entered as a percentage of the subdivision's duration. Else, SMS Conductor will only take queues within the deviation limits surrounding the measure's downbeat. In this case, the deviation percentage refers to the measure's duration.

If a measure (generally a group of measures) is marked *Ad libitum*, SMS Conductor will place a fermata at the end of the group. Time stretch will not be applied during this section. A conductor's queue will then position the application in alert state waiting for a second queue to start playing the measure after the fermata. If the section was not done yet, SMS Conductor will jump to the fermata location and skip part of the sound. If the section has been entirely played, three options can be applied:

1. The synthesis is paused waiting for a queue to continue.

2. The synthesis is paused, a short amplitude decay is applied to the synthesis before pausing, and a short amplitude rise applied when restarting.

3. An endless loop is applied on the synthesis of the last few frames of the analysis waiting for a queue to move forward.

These options are selected by using one of three different types of fermata. (*Figure 6*)

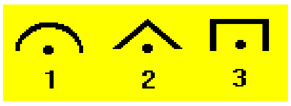


Figure 6

3.3 Other score editing options

The length of the drawn squares representing the measures is proportional to the measures duration. When the user enters new tempi or accelerandi/ritardandi markings the duration of the measures and their graphical representation are modified accordingly.

Besides tempo markings, fermata and text, dynamic markings can also be entered (crescendi, decrecendi, *forte*, *piano*, etc.). When entering dynamics, the user can either select to enter them simply as reminders (graphical display) or allow them to actually modify the synthesis amplitude. In this case, the user must define the value in decibels of the dynamic symbol he chooses to display on screen.

Finally, rehearsal sections can be defined and linked one to another in case a similar or identical measure properties and distribution need to be shared at different places in the music (reexpositions). When rehearsing, the conductor can easily move to the beginning of any of the defined rehearsal sections and start performance at that point.

3.4 Synthesis transformations

In addition to the conductor's guided time-stretch fluctuations, SMS Conductor allows the user to predefine envelopes to further modify tuning (frequency), amplitude, and stretch/compress parts of the original sound (*Figure 7*). This envelope transformation can be saved, if wished, as a new SMS analysis file, or added at performance time.

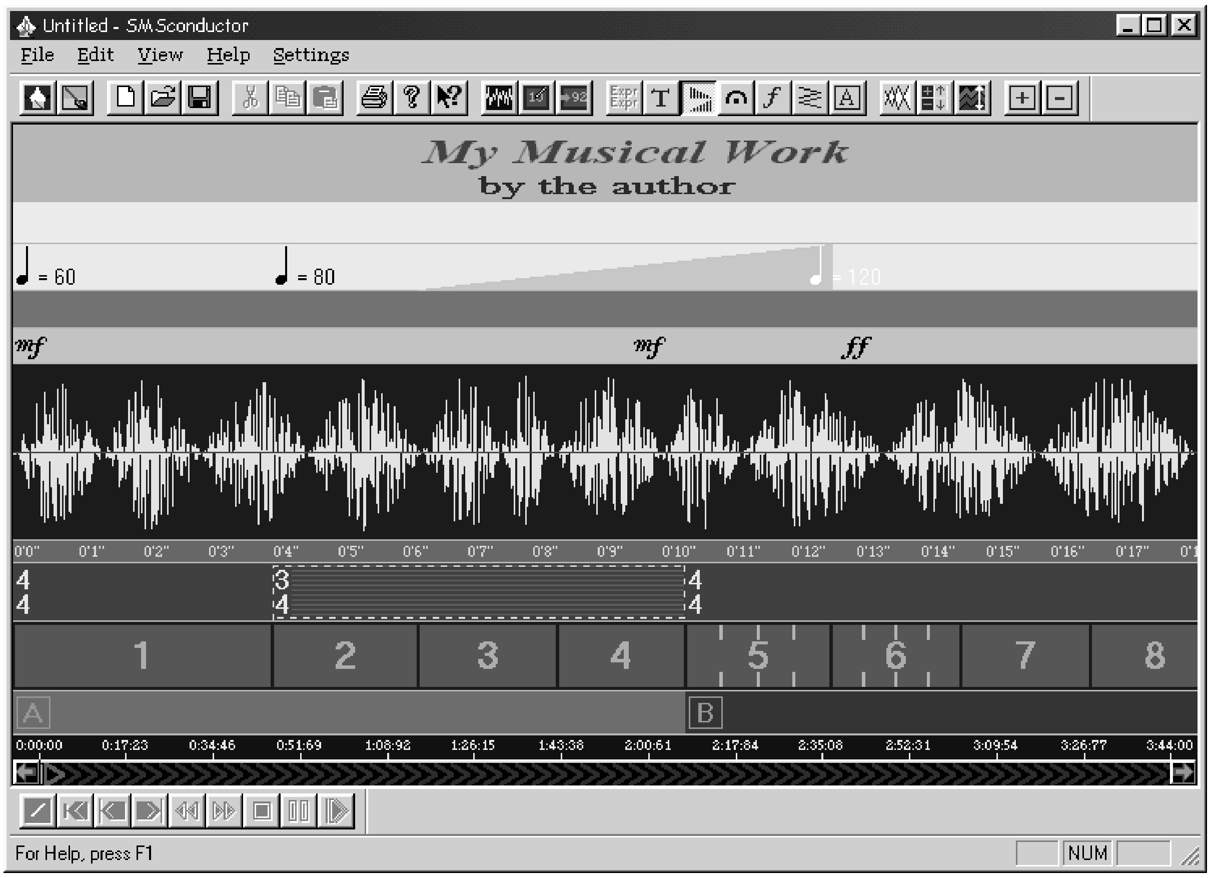


Figure 7

The latter parameters can also be modified during performance through sliders that are displayed at the top of the screen when SMS Conductor is in performance mode, or by means of a set of MIDI continuous controllers (MIDI table). There is a fourth slider to set the expected reactive delay of the musicians to the conductors actions (which varies depending on the country, the conducting style, the music, the musicians, and even the acoustics of the hall), in order to prevent the synthesized sound to anticipate to the musicians playing.

3.5 On conducting

SMS Conductor will wait for and measure two to four queues (as selected) prior to starting to play. This will set the initial tempo as long as the first measure allows deviations on the expected tempo, otherwise the expected tempo will be taken. From this point on, SMS Conductor will collect the MIDI_IN messages. If there are no input messages the application will stand by the expected tempo. When a message gets in, the expected musicians reactive delay is added to the message's arrival time. If this added time falls within the time margin set by the current beat or measure's deviation, the message will be processed. SMS Conductor does not only measure the incoming messages against the expected tempo, but also takes in account the last established tempo which attempts to preserve.

For example: if the score is supposed to accelerate from a tempo of 60 to a tempo of 80, and the conductor is already conducting at 80 (translated to a time stretch synthesis value of 1.33333), SMS Conductor will try to accelerate accordingly to a tempo around 106, it will not stay at 80 (which would mean to fall back down to a stretch value of 1). If the allowed deviation at that point is smaller than the attempted one, SMS Conductor will accelerate as much as possible within the given margin as long as no new conductor's queues are detected.

The messages are processed by a separate thread which sends the transformation data to the synthesizing computer(s) via MIDI. This is done this way because, even though MIDI_OUT messages need not be sent with a frequency higher to three or four times per second (unless envelope values are applied during performance, in which case messages are sent at a frequency of 100Hz), SMS Conductor must be free to receive and evaluate incoming messages at a much higher frequency (both MIDI and system messages).

During performance, the conductor's tempo is monitored as a level fluctuation above and below the expected tempo. Two clocks showing score time and performance time appear on screen as well as the current measure number and tempo numeric value. There is also a sort of street light indicator that lets the conductor know whether or not his/her action has successfully reached its goal.

3.6 The MIDI interface

Any device capable of sending a MIDI note-on can be used, though hopefully we will avoid having a conductor pressing a keyboard key with one hand while conducting with the other.

The interface we are currently experimenting with is GAMS [8]. This device uses four speakers, conveniently located, that send ultrasounds by means of which, the position of a small interface (wand) placed on the conductor's hand is detected. Even though the device is capable of tracking, not only the position of the hand, but also the speed and direction of its motion (which could inspire a more complex and risky attempt to guessing the conductor's intentions), it is safer not to ask a conductor to substantially modify his/her usual behavior.

Future versions of SMS Conductor will include the possibility of entering a musical score representation or a midi file to track and match in performance, allowing the application to synthesize the sound following a musician's playing which will be sending note messages via a MIDI pitch converter.

4 Conclusion

SMS has, for some time now, been developed and researched with few applications to composition and performance. The musical practice world imprints a series of demands that are currently being translated into user applications, which will allow musicians to take advantage of the great potential that SMS offers them in their various creative tasks. SMS Composer and SMS Conductor are two of such applications intended to enrich the possibilities of both computer generated works and computer assisted performance.

5 Acknowledgments

SMS Composer uses the SMS Classes written by Jordi Bonada to synthesize SMS files.

I want to thank all the research team working on SMS at the Audiovisual Institute of the Pompeu Fabra University in Barcelona, and especially to Xavier Serra and Jordi Bonada for their help and support.

References

[1] X. Serra. "Musical Sound Modeling with Sinusoids plus Noise". G. D. Poli and others (eds.), *Musical Signal Processing*, Swets & Zeitlinger Publishers, 1997.

[2] A. Loscos and E. Resina. "SmsPerformer: A Real-Time Synthesis Interface for SMS". *Proceedings of the Digital Audio Effects Workshop (DAFX98)*, 1998.

[3] E. Resina. *L'Esquizofrènia dels Sons*, for reciter, septet, and computer generated sound, 1993-94. (SMS running on NeXT).

[4] E. Resina. *si eSe Me vieSe*, 1996. (SMS running on NeXT).

[5] E. Resina. *Menstruació*, for female voice and computer transformed voice, 1998. (SMS on Windows).

[6] J. Rahn. "The Lisp Kernel: "A Portable Software Environment for Composition", *Computer Music Journal* 14(4):42-58, 1990.

[7] H. Taube. "Common Music: A Music Composition Language in Common Lisp and CLOS", *Computer Music Journal* 15(2):21-32, 1991.

[8] GAMS by Acoustic Positioning Research Inc.