METRIX: A Musical Data Definition Language and Data Structure for a Spectral Modeling Based Synthesizer

Xavier Amatriain, Jordi Bonada, Xavier Serra
Audiovisual Institute, Pompeu Fabra University
Rambla 31, 08002 Barcelona, Spain
{xamat,jboni,xserra}@iua.upf.es http://www.iua.upf.es

Abstract

Since the MIDI 1.0 specification [1], well over 15 years ago, many have been the attempts to give a solution to all the limitations that soon became clear. None of these have had a happy ending, mainly due to commercial interests and as a result, when trying to find an appropriate synthesis control user interface, we had not many choices but the use of MIDI. That's the reason why the idea of defining a new user interface aroused. In this article, the main components of this interface will be discussed, paying special attention to the advantages and new features it reports to the end-user.

1 Introduction

A lot has been written about the problems in MIDI: speed, channels, program numbers, modes, stuck notes...

The new proposals which are more likely to be accepted by the synthesizer industry are those which not only keep compatibility but also are just extensions of MIDI such as X-MIDI [2] or Fuzzy-MIDI [3]. These possible standards offer different improvements to MIDI limitations but don't face the problem for which MIDI doesn't seem a long-term supportable standard : keyboardism.

The basics of MIDI start tumbling when it comes to controlling a string synthesizer, which can have two strings playing the same note but sounding different, or a wind synthesizer, which can have continuos controls such as lip pressure or frequency.

The solutions given to this problem start by being able to control not only key numbers, instruments or channels. Some proposals such as Open Sound Control [4], Synth Control [5] or SKINI [6] offer the possibility of controlling any kind of synthesis events by giving them their own address or ID number.

A more complete solution was given by ZIPI [7]. In ZIPI, an instrument was described as a set of *notes* which could be addressed and modified independently. This concept is similar to the one we have named *generator*, the minimum unity within an instrument which is capable of producing a sound by itself (a string, a key, a drum...). Needless to say that this idea of *generator* has little to do with the idea of *synthesis generator* used by some protocols as the Soundfont 2.0 [8].

The whole system, is a transport-independent protocol, and although it was not designed with a particular transport layer in mind, features shared by modern networking technologies are assumed. Therefore, the design is not preoccupied with reducing musical information to the minimum.

2 General Concepts

The interface is made up of a Musical Score Description Language (MSDL), an SMS [9] Instrument Definition Language (SMSIDL) and a set o C++ classes which together with other more general purpose classes conform the SMS Class Structure (SMSCS)[10]. The MSDL is intended to be general purpose and might be useful to other synthesis techniques while the SMSIDL and the SMSCS are based on our particular needs. The distinction of an Instrument File from the Score File has already been used in synthesis control system being the most renamed Csound [11].

The goal of the complete interface is to offer an easy-to-use but yet flexible tool to control all the different aspects involved in digital instrument synthesis. In the following sections I will give a brief description of its main components.

3 The Spectral Modeling Synthesis Class Structure

In this section, only the classes in the SMSCS that are more related and have indeed been developed for this layer are discussed.

Up to this moment, no compatibility has been intended with MPEG-4's Structured Audio

Specifications (SAOL) [12] although the main guidelines have been observed.

2.1 The Synthesis Controller Class

This is, in fact, the class responsible for controlling all the different aspects involved in the synthesizer.
Its first commitment is to read the information contained in the Instrument File and in the header of the Score File initializing all instruments and global variables that will later be involved in the synthesis process.

Then, it starts and keeps control of the Synthesis Loop in which it receives new events from the Score File and controls the Generators active at that moment. The Synthesis Controller keeps trace of active Generators and Instruments.

global Synthesis Classes necessary and sufficient to fill a synthesis buffer by themselves.

Each Generator has also a pointer to the Instrument Class in order to obtain all the information loaded from the Instrument File and that affects that Generator.

Another important feature contained in each generator is a pointer to a Synthesis Parameters Class. This is a special class that keeps trace of all the different synthesis parameters (low and high-level) and their value at any time.

2.4 The Score File Class

The Score File Class is the class responsible for reading the information stored in a standard text file

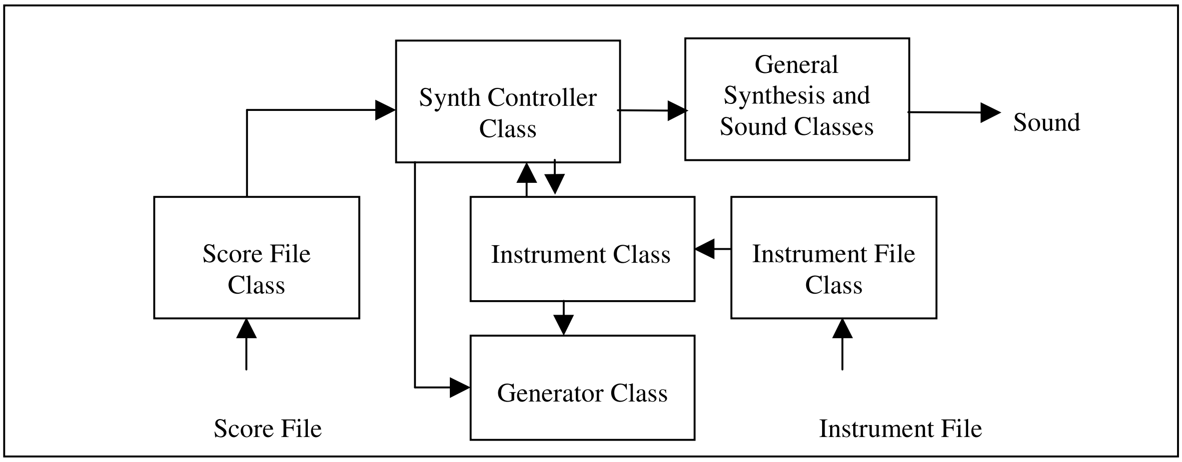

Figure 1. Class Structure.

according to the MSDL general rules. The class reads all the information contained in the Header as well as all the events when its member function Load is called.

When the Synthesis Controller Class initializes all instruments and generators, it asks for the information loaded from the Header of the score file. Then, in the synthesis loop it keeps asking for the events with a time tag included within the current synthesis period until the Score File Class recognizes the end of the file being read. It is clear that this kind of working is fully compatible with the real-time idea of reading events stored in a buffer since the last call from the Synthesis Controller.

2.2 The Instrument Class

The Instrument Class contains all the information read from the Instrument File.

One of the main features in the Instrument Class is the Timbre Space. Although the name and concept has been taken from previous works [13] its purpose and implementation is completely different. The Timbre Space is an n-dimensional virtual space formed by the positioning of the different SMS Data Tracks in a specific location. The class knows how to obtain the appropriate Track Frames by interpolating the already loaded information. See section 4.2 for more information on the Timbre Space.

2.3 The Generator Class

As mentioned in section 1, a Generator is the minimum unity within an instrument capable of producing a sound by itself, and that is the reason why the Generator Class has pointers to all the SMS

2.5 The Instrument File Class

This class is responsible for reading all the information contained in a standard text file according to the SMSIDL. After reading, all the information is loaded in a special structure accessible from the Instrument when the Synthesis Controller asks for its initialization.

3 The Spectral Modeling Synthesis Instrument Definition Language

No low-level bit structured messages are involved in defining an Instrument File with the SMSIDL. A set of reserved words are defined which combined, following an easy syntax, conform the messages in an Instrument Definition File. An SMSIDL File can be a standard ASCII text file and be modified with any word processor or either any kind of real-time stream.

The Instrument File is divided into four different parts that conform the complete definition of an SMS based Instrument, that is: definition of Instrument Generators, Instrument Timbre Space, Time and Parameter Envelopes and SMS and Control Parameters. All of them will be introduced in the following sections. A simple example is given in section 4.5.

It should be pointed that the order of these parts in the Instrument File is indeed critical and must follow the one here introduced. Otherwise, references to not previously introduced information, will not be solved.

2.6 Generators

In this part of the Instrument File, a unique name and integer identification number must be given to each generator.

The identification number will be used in the other parts of the Instrument File in order to access the different generators. The name will be accessed later from the Score File.

2.7 Timbre Space

The idea of a Timbre Space was already introduced in section 2. The definition of a Timbre Space with the SMSIDL consists in the definition of three different aspects.

First, the number of dimensions to be used. The instrument designer must decide what features of the instrument represent a substantial change in the sound that cannot be achieved using the different transformations available. These dimensions such as loudness, pitch, articulation, etc... must have the correspondent SMS data extracted from a previous analysis of those features in the instrument. A compromise between sound quality and amount of synthesizer memory used must be adopted.
Next, the kind of Interpolation to use between the Data stored. A set of standard interpolation types are available.

Finally, the positioning of each SMS data in a concrete location in the space.

The Instrument Class will be capable of solving intermediate positions by the interpolation of the SMS data loaded from the files.

E.g. A good quality piano sound can be obtained by storing the SMS Data from just eight of the piano keys (A0 thru A7) and obtaining the rest by interpolation. Thus, only one dimension is used (pitch). Other features such as loudness can be obtained by applying different transformations on the data available.[14]

2.8 Envelopes

Two kind of envelopes are available: Time Envelopes and Parameter Envelopes. An envelope is defined by giving a name, an interpolation type and any number of envelope points from which the others will be computed.

A Time Envelope is a user defined function that returns a single value according to the relative time elapsed since the beginning of an event.

Unlike Time Envelopes, Parameter Envelopes return a complete envelope that will be applied to the parameter involved according to its definition.

2.9 Parameters

In this part, SMS and Control Parameters are initialized. Other parameters, will not be accessible from the Score File.

To initialize an SMS Parameter, only its maximum, minimum, and default value are specified.

Besides that, when initializing a Control Parameter, the instrument designer must define the relationship between that parameter and the set of SMS Parameters or the Timbre Space location. A single Control Parameter can have any number of low-level references. Any kind of standard formulas or Envelopes previously defined in section 4.3 may be used in this field. An standard set of Control Parameters is available but the list is still not complete as it depends on each kind of instrument to define and the musicians gestures to describe.

2.10 Example

In the following example, a simple SMS based piano synthesizer is implemented [14]. Only one dimension of the timbre space is used and the number of SMS and Control parameters used have been reduced to the minimum to keep the example simple. A more complete explanation is given at [10].

The reserved characters '#' and '@' mean the value of the parameter and the number of the generator involved respectively.

```
Generators :{
     "Name[0-85],Key" }
SMSTimbreSpace :{
     "nCoord,1"
     "IntType,PianoPitch "
     "c:\Metrix\Piano\A0.sms,0 "
     "c:\Metrix\Piano\A1.sms,0.1412 "
     "c:\Metrix\Piano\A2.sms,0.2824 "
     "c:\Metrix\Piano\A3.sms,0.4235 "
     "c:\Metrix\Piano\A4.sms,0.5765 "
     "c:\Metrix\Piano\A5.sms,0.7176 "
     "c:\Metrix\Piano\A6.sms,0.8588 "
     "c:\Metrix\Piano\A7.sms,1 " }
ParamEnvelopes :{
 "PianoLPF,Linear,   (0,1)(0.5,0.39*# +0.5)(1,0.00006*# ^2)" }
TimeEnvelopes :{
     "FadeOut,Linear,(0,1)(T,1)(T+0.3,0.1)(T+0.6,0)" }
SMSParams :{
     "Amp,0,0.2,0.1"
     "AmpSine,0,1,1"
     "AmpSpec,0,1,1" }
ControlParams :{
     "KeyVelocity,0,127,64,
     [Amp,#/127*0.2*TimeEnvelope(FadeOut)]
     [AmpSine,ParamEnvelope(PianoProva)]
     [AmpSpec,0.0000506*#^2+0.00145*#]"
     "KeyNumber[@],@,@,@,[TimbreSpace,(@/85)]"
     "Pitch[@],28.8316*1.0595^@,27.16*1.0595^@,28*1.0595^@
     , [TimbreSpace,(@/85)]" }
end
```

3 The Musical Score Description Language

The MSDL is a text-based synthesis control language which takes part of its features from previously released languages as the NEXT ScoreFile Language [15] or SKINI [5]. No low-level packed messages are involved in defining a Score with the MSDL. With a quick look at an MSDL Score File any musician can get a grasp of what is going on. A simple example is given in section 5.3.

As the Instrument Definition File, a Score can be a standard ASCII text file and be modified with any word processor or either any kind of real-time stream.

Although no exact match with the SMSIDL syntax is meant, similarity and compatibility is intended.

The Score is made up of two different parts, which will be discussed in the following sections: the Header and the Body.

2.11 The Score Header

The Score Header is where all the global variables relative to the score information or to the output sound are defined. Concepts such as Tempo, Beat, Output Sound File or Sample Rate must be included in this part or either will be assumed as default.

Another feature included in the Header is the definition of all the instruments to be used in the score. A reference to the Instrument File location must therefore be included.

And last but not least, all kind of user variables can be initialized in this part of the Score. The user can define an unlimited set of variables in order to access the instruments, generators or even groups of instruments. Note that if more than one instrument of the same kind is to be used, its Definition File will only be loaded once and can then be referenced by the use of user defined variables such as *piano1, piano2...*

2.12 The Score Body

The Body is the part of the Score where the actual musical information to control a digital instrument is included. It is made up of a list of events; sorted by the time they take place in order to keep real-time compatibility.

An event is a group of words that define a message sent to the synthesizer controller. The standard event statement is made up of four statements sorted this way: *T V P:PV,* where *T* is *Time Statement, V is Variable Statement, P* is *Parameter Statement,* and *PV* is *Parameter Value Statement.*

These, together, conform a message that means: Modify Parameter (P) referring to variable (V) according to its new value (PV) at the moment specified (T).

The Time Statement includes features such as the possibility to use standard time, SMPTE timecode or musical Beat notation.

Variable statements can refer to user defined variables or directly to instruments initialized thus affecting all variables.

There are two levels of indirection, which the user can access from the Score in order to control an SMS based instrument. These two levels correspond to the two kind of parameters that can be used in the Parameter Statement: the low-level SMS Parameters, or the high-level Control Parameters. The SMS Parameters control instrument features such as the amplitude of the partials and transformations such as pitch shifting or morphing. There is a complete list of parameters at [10].

2.13 Example

This example shows the main possibilities of using the MSDL for controlling a synthesis process. Note the different kind of instruments used: the first two have already been defined and included in the synthesizer standard bank, the next two are loaded from an Instrument Definition File during run-time, and the last one is a single SMS File containing any kind of information accepted by the SMS File Format [10].

```
Score_Info{
   Tempo:130/2
   Meter:3/4
   Resolution:24 }
Sound_Info{
   Bits:8 }
Instrument_Info{
   Piano
   guitar
   oboe[InsDef:"c:\score\oboe"]
   violin[InsDef:"c:\mpegscore\violin"]
   clarinet[SMSDef:"c:\SMS\clarinet"]}
Def instrument a=piano
Def instrument c=violin
Def generator nvar=10 c=c.string
Def instrument d=violin
Def generator a1=a.key1
Def generator a2=a.key2
Def instrument n=clarinet

begin

#01:01:02.04 a1  Pitch:C#3 Loudness:mf  Duration:t00:00:01
t00     clarinet      AmpFn:[0(1)1(0.5)]
t04     piano.key2    Pitch:f2
t04     Score_Info:   Tempo:140
t05     a1            Loudness:ffff

end
```

3 Conclusions

The whole interface is meant to be simple but yet flexible enough to offer a complete set of classes and syntax rules that could be enhanced in the future to observe other features that have not been included in this first implementation. MIDI and other interfaces compatibility is also thought to be available in the near future.

Up to this moment, the program is running as a stand-alone utility for PC but it will soon be included in a more general SMS interface.

Definitions of other instruments as well as other score examples are also on the way. All new features will be available at [10].

References

[1] MIDI Manufacturers Association. *MIDI 1.0 Detailed Specification*. Los Angeles: The International MIDI Association, 1998.

[2] E. Lukac-Kurac. *Extended Midi White Paper*. Meise, Belgium: Digital Design & Development, 1995. ftp://ftp.cs.ruu.nl/pub/MIDI/DOC/xmidi.html

[3] S. Wilkinson. "Fuzzy MIDI (MIDI Spec Revision)". *Electronic Musician*, April 1995.

[4] M. Wright and A. Freed. "Open SoundControl: A New Protocol for Communicating with Sound Synthesizers". *Proc. ICMC, 1997.*

[5] M. Wright and A. Freed. *The Synth Control network protocol version 1.0*, 1996. http://cnmat.cnmat.Berkeley.edu/Adrian/SynthControl.html.

[6] P. Cook. *Synthesis toolKit Instrument Network Interface (SKINI) 0.9 Implementation notes*. Princeton University, 1996. http://www.cs.princeton.edu/~prc/SKINI.txt.html

[7] K. Mc Millen. "ZIPI: Origins and Motivations". *Computer Music Journal* 18(4), pp 48-96, 1994.

[8] E-mu System Inc. *Soundfont® Technical Specification, Version 2.00a*, 1995.

[9] X. Serra and J. Smith. "Spectral Modeling Synthesis: A Sound Analysis/Synthesis System based on a Deterministic plus Stochastic Decomposition". *Computer Music Journal*. 14(4). pp12-24, 1990.

[10] Music Technology Group. *SMS Homepage*. http://www.iua.upf.es/~sms

[11] B. L. Vercoe. *The CSound Manual Version 3.48. A Manual for the Audio Processing System and supporting program with Tutorials*. MIT Media Laboratory. Edited by Jean Piché, University of Montreal, 1992. ftp://ftp.musique.umontreal.ca/pub

[12] Synthetic/Natural Hybrid Coding (SNHC) section of the MPEG-4. *Final Committee Draft Version 1.8. Document num.FCD ISO/IEC 14496-3 Subpart 5*. MIT Media Laboratory, 1997. http://sound.media.mit. edu/mpeg4

[13] D. L. Wessel. "Timbre Space as a Musical Control Structure". *Computer Music Journal*, 2(3), 1979.

[14] J. M. Solà. *Disseny i Implementació d'un sintetitzador de piano.* Graduate Thesis. Polythecnic University of Catalonia, 1997.

[15] E. Selfridge-Field. *Beyond Midi*. MIT Press, 1997