A GRAPHICAL MODULAR ENVIRONMENT FOR MPEG-4 SA INSTRUMENTS PROTOTYPING

Guillaume Fayemendy

France Telecom R&D
DIH/HDM
4 rue du Clos Courtel, 35510 Cesson Sevigne Cedex, BP59
guillaume.fayemendy@rd.francetelecom.com
guillaumefy@free.fr

ABSTRACT

This paper presents current work being made in the development of an ergonomic graphical environment for MPEG-4 SA instruments creation and manipulation. The project goal is to define a means of giving easy access to the power and flexibility of the MPEG-4 SA framework.

1. INTRODUCTION

Structured Audio is a sound synthesis framework defined in the MPEG-4 standard. Its main part is the Music-N model based Structured Audio Orchestra Language (SAOL). We enlighten some roadblocks to the use of this tool and propose a software application giving an easier access to it. After a general description of this application, we will present some aspects of the architecture model it relies on.

2. MPEG-4 STRUCTURED AUDIO

2.1. Presentation

Structured Audio is a component of the MPEG-4 standard framework for sound synthesis. It allows for the efficient and flexible description of synthetic music and sound effects, and the use of synthetic sound in synchronisation with natural sound in interactive multimedia scenes [1]. It's been NetSound [2], sound and music specification protocol oriented towards networked low-bandwidth sound synthesis applications, which led Vercoe and al. to formalise the Structured Audio representations concept [3]. These are description formats that are made up of semantic information about the sounds they represent and that make use of high-level or algorithmic models.

At the heart of the MPEG-4 SA toolset is the Structured Audio Orchestra Language (SAOL)[4], a music synthesis and effects processing language based on the Music-N model. It enables the efficient and flexible description and transmission of synthesis processes controllable by MIDI instructions and/or by the lightweight Structured Audio Score Language (SASL). It involves a new format for transmission of samples banks dedicated to wavetable synthesis and called SASBF [5].

2.2. The SAOL Structured Audio Orchestra Language

SAOL is the latest incarnation of the music-N software model. It is a declarative unit-generator based language, unit-generators being primitive modules for generating, modifying, and acquiring audio or control signals (they are called ”opcodes” in SAOL). Instruments for sound synthesis or effects are obtained by connecting unit-generators, and an orchestra is a collection of instruments. SAOL extends the syntax of Csound, one popular predecessor (for which many informations can be found at [6]) in order to make it more understandable and concise. It adds a number of new features to the music-N Model. Its design goals have been high modularity, expressiveness, and functionality : anything that can be done with digital audio could be expressed in SAOL and produced by a standard desktop computer.

SAOL retains many well established features of music-N languages : the sample-rate/control-rate distinction, orchestra/score distinction, the use of instrument variables, global variables and stored-function tables. It allows the use of signals and unit-generators arrays and introduces dynamic extension capabilities of the unit-generators set built into the specification by defining user-opcodes without requiring to rebuild the language system. It also uses a metaphor of the mixing console with ”send” and ”return” audio busses, which serves as a stylish means of expressing chains of sound effects. Its weaknesses are clear when looking at the basic model for MIDI control, the distinction between orchestra and score, and the lack of formalization for object oriented approaches.

The reader interested in a presentation of the SAOL language relating to concurrent music programming languages and other software applications devoted to digital audio could refer to [7] in which aspects regarding to modularity, expressivity, design issues, performance, extensibility and capabilities to deal with audio effects are discussed.

3. MOTIVATIONS

Despite its power and flexibility, it remains that SAOL is a programming language that requires specific knowledge and therefore has a long learning curve (although it uses a ”C-like” syntax which is quite legible). This prevents it from being widely adopted by musicians and sound designers who may not be particularly inclined to learn all the intricacies of the language. The aim of this work is to provide an environment, based on graphical components, allowing a user to set an orchestra and to create an output compatible with any compiler conforming to the MPEG-4 Structured Audio format. It should give the user a way to perform experiments with multiple instruments and give him an access to the most expressive parameters.

3.1. Working with music programming languages

Numerous difficulties have been revealed by users of previous music programming languages such as Csound (this language has been around for a long time and has developed a large community of users).

Even for the user who masters a language, some difficulties arise :

1. the necessity to gather all the resources needed. These being text editor, sound file player and editor, a conform decoder...
2. the inconvenience of having to use text to describe time-based functions.
3. text score generation : in MPEG-4, SASL language enables the definition of a collection of time-stamped invocations of instrument events in text form. Utilities for graphical or high level handling are needed.
4. one aspect that is particularly clear with Csound is the difficulty to re-use code for a different purpose it has been designed for. Yet there's a large number of instruments that have been designed by fellow sound designers and which are freely available. With the numerous parameters involved in the definition of an instrument, it can be difficult finding the most significant and appropriate ones to use and to modify for a different context.

3.2. Front-Ends

In the Csound world, several graphical helpers applications have been proposed. They provide a user interface to the sound synthesis engine. They aim to propose alternatives to some of the more tedious tasks involved in making sounds with programming languages, and they produce reasonably user-friendly capabilities.

Some of these applications rely on the graphical programming approach initiated by Miller Puckette [8] with the MAX package. Others may involve no programming but will be limited to the use of a particular dedicated sound synthesis method. Therefore, much effort in the handling of score production oriented tasks have resulted in useful tools for classic music representation/manipulation or algorithmic composition (see [9] for example).

A widely recognized complete production system is Cecilia [10], which makes use of the Csound language biased towards the production of sound-object oriented composition and offers a high level language for complex score production.

When looking at SAOL and MPEG4-SA, we must recognize the efforts of the ”saol.net” team [11] and the Snet text editor application dedicated to SAOL. It's a simple editor that includes such features as syntax coloration, embedded decoder, and wave player.

The aim of our work is to propose an optimal environment focusing on the manipulation and creation of instruments in the MPEG4-SA context. We hope to establish an efficient relationship between the textual nature of the programming language and the creation of graphical user interfaces (the system proposed isn't merely another graphical patch editor in the MAX style). Our approach to this problem has been entirely led by the interaction capabilities needed for both novice SAOL users and experienced ones. We are not investigating aspects of music composition but we shall propose a simple graphical tool derived from the classical piano-roll representation to provide a means of adding synthetic materials in a real musical context.

4. ENVIRONMENT DESCRIPTION

As mentioned earlier, both novice and expert SAOL users are to be catered for. Novice users will play with and modify predefined instruments, with all the intricacies of the language being hidden. For SAOL programming language experts, the environment provides facilities for the creation of new instruments. The idea is to propose an environment which enables the description of a user interface when designing a new instrument. The graphical user interface must be efficiently modular to adapt itself to the variety of possible sound generation algorithms. The expert user will have to put in evidence the parameters conveying the most relevant information for edition and manipulation.

For this purpose we shall consider an instrument as the sum of agents contributing to its entire definition. ”Agent” belongs to the vocabulary of the underlying software architecture model we rely on. The latter will be detailed in section 5.
The principal agent (called ”core-code element”) is a classic text element describing partially, in the SAOL syntax, a sound generation algorithm. Other agents are graphical elements conveying an abstraction of an opcode. Their shared role is to produce code depending on their graphical state. The agent's contribution to the whole code production of the instrument is dealt with a supervising agent.

4.1. Interaction elements

Of the hundred unit generators built into the SAOL specification, there are some that are biased towards to a (reasonably standard) graphical representation and provide a good set of elements for interaction. Good examples are the ”kline” generator or the wavetable ”lineseg” opcode which handle the abstraction of the break-point function, very useful for temporal variable definition.
In its elementary form, an agent will deal with a single SAOL variable. On request, it can provide the code for its declaration or for its initialization.
We now need to identify the different types of elements that compose a SAOL instrument :

- Parameter fields (these being the variables defined in the score by the SASL language)
- Initialization rate variables (these are represented by a single fader)
- Control rate variables
- Wave tables variables
- Core-code element

This is currently being research into : all generators in the elementary set that prove inclination in being graphically interfaced are not yet implemented and the direct interfacing of selected user-opcodes is being looked into. For example a user-opcode defining a common recurrent element such as LFO (which is not defined in the set of elementary unit-generators) should be chosen and interfaced with a graphical panel allowing the specification of the waveform used, the frequency, and of the speed.

4.2. Instrument creation and manipulation

An instrument window contains all of the elements involved in the composition of an instrument. It's split in two horizontal panels (See Figure 1). The panel at the bottom is devoted to the construction and definition of the instrument. It's not naturally visible and its different parts can be edited individually. It's where the expert user will choose and parameterize the graphical elements, define the core-code element by writing the SAOL code in the text editor, and defines the parameter fields of the instrument.
The panel above is the control panel where graphical elements take place. Buttons for selecting the elements/variables to edit are vertically aligned according to their types (initialization/control/wavetable).

A snapshot mechanism permits the user to save and reload any particular instrument configuration.

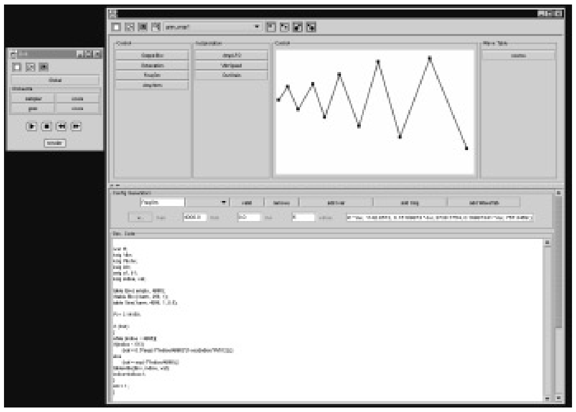

Figure 1: *Edition of a simple granular instrument.*

4.3. Score management

A simple tool (figure 2) has been designed and implemented to put instruments in a real "musical" context. It's derived from the traditional piano-roll representation. Event representations, made of rectangular symbols of assorted lengths, with colours ranging from white to black, are added and transferred into the grid with the mouse. The resultant difference from piano-roll resides in the capability to choose which variable of the parameter fields will be assigned to the vertical axis (generally attributed to pitch), to the length of the event representation (which can be used to set a parameter for percussive sounds), and to its colour.

At the bottom of the score window, a panel of faders representing the whole parameter fields for an edited event is used to complete the score definition. The handling of import variables is undergoing study.

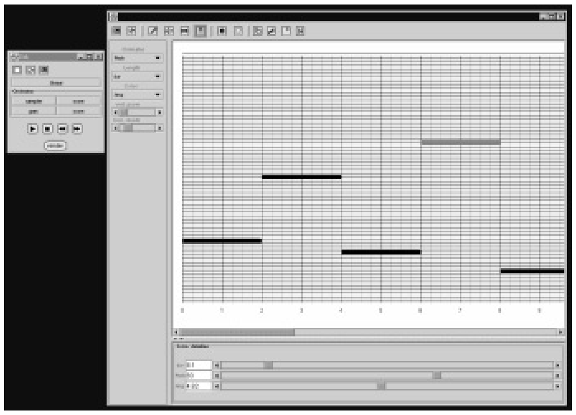

Figure 2: *A graphical tool to produce little scores.*

4.4. Orchestra management

The whole environment is composed of two windows that allow intuitive and clear navigation. One can select an instrument from the orchestra's instruments list to edit or to identify its score counterpart. For the moment, however, only instruments used in conjunction with the SASL language are available. A few modifications to the environment for the handling of MIDI instruments and effects instruments need to be carried out.

5. SOFTWARE DESIGN ISSUES

5.1. The PAC (Presentation Abstraction Control) architecture

We constructed our software using the PAC architecture model. It's a design abstraction introduced by Coutaz [12] for the software conception and development of interactive systems. This model defines a system as a collection of specialized computational units called agents. Those agents feature facets that are used to express different yet complementary and firmly paired computational perspectives of the same entity. These facets are the Presentation, the Abstraction and the Control. This architecture model extends the "separation of the functional core from the user interface" principle.

5.2. The PAC agent

The PAC agent has three facets which are :

- the Presentation defines the system image, i.e. its perceivable input and output behavior;
- the Abstraction defines its functional core;
- the Control is in charge of communicating with other agents, as well as expressing dependencies between the Abstract and Presentation facets of the agent.

Note that the Presentation facet can be decomposed in multiple elements and thus permits to expose a concept with multiple views.

5.3. PAC'ing a SAOL opcode

We describe the "kline-opcode" agent (see Figure 3). Its first presentation is a rendering surface to set break-point function values.

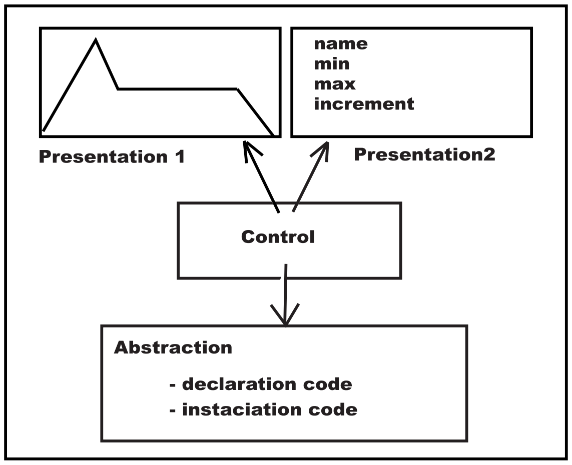

Figure 3: *"Kline opcode" agent.*

Its second presentation is used to define the settings (like minimal and maximal values).

The abstraction facet of the "kline-opcode" will handle the data needed for the SAOL code production.

The control's role is to ensure the coherence between the abstraction facet and the presentation facet. The abstraction facet with its specific semantic, deals with data expressed in SAOL while the data of the presentation are expressed relatively to the graphical referential.

5.4. Agents organization and code production

An agent is a unit of competence which operates in parallel and in coordination with other agents by the mean of their Control facet. They are organized in a hierarchical tree, a parent agent having the responsibility of its children (Figure 4).

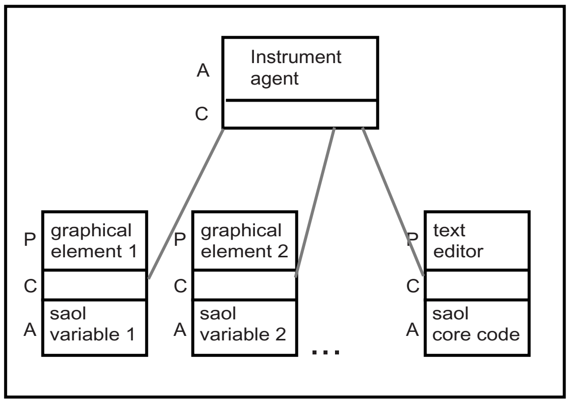

Figure 4: *The instrument agent and its children.*

Some agents have neither abstraction nor presentation facet. In our application, it's the case of the instrument agent for which the presentation is the sum of its constituents graphical's parts. It's then defined by its abstraction and its control, this one ensuring coherence between its children (mainly the graphical SAOL variables relative to an opcode's abstraction and the core-code element).

The code is produced in a simple way according to the SAOL syntax. At the instrument level :

- in a first pass, the instrument agent collects from its children the code necessary for the variables declaration,
- in a second pass, it collects the code necessary for the graphical variables initialization according to their state and appends the core-code element's contributing part of code.

6. CONCLUSIONS AND PERSPECTIVES

We have studied, in the light of actual software applications dedicated to digital audio, some difficulties relevant to the work with music programming languages. Then we have proposed a front-end application to the SAOL language relying on the simple but robust object-oriented PAC architecture. Its strengths are its multiple views capabilities, and the re-usability and evolutivity of the application components. It should be simple to modify the application for other music programming languages.

The whole implementation has been made in java and the XML format is used to handle instruments and score data. Some part of code for the score utility has been derived from a little original open source java application dedicated to Csound [13]. Our application is powered up by the Sfront SAOL decoder [14].

The development is in working progress but the application reveals to be particularly responsive, and seems to really provide friendly ways for experimenting with sound synthesis. One next step will be the integration of sound analysis tools.

7. REFERENCES

[1] E.D. Scheirer, "The MPEG-4 Structured Audio Standard", IEEE Trans. Speech and Audio Proc., 1998.

[2] M.A. Casey, P. Smaragdis, "Netsound" Proc. ICMC 1996, Hong Kong.

[3] B.L. Vercoe, W.G. Gardner, E.D. Scheirer, " Structured Audio: Creation, Transmission, and Rendering of Parametric Sound Representations ", Proc. IEEE 86:5 (May 1998), pp. 922-940 (invited paper).

[4] E.D. Scheirer, "The MPEG-4 Structured Audio Orchestra Language", Proc. ICMC, 1998.

[5] E.D. Scheirer, L. Ray, "Algorithmic and Wavetable Synthesis in the MPEG-4 Multimedia Standard", 105th convention of the Audio Engineering Society 1998, San Francisco, Calif.

[6] http://csounds.com

[7] N. Bernardini, D. Rochesso, "Making Sounds with Numbers: A tutorial on music software dedicated to digital audio", Proc. COST G-6 DAFX, 1998.

[8] http://crca.ucsd.edu/ msp/index.html

[9] M. Gogins, "Music Graphs for Algorithmic Composition and Synthesis with an Extensible Implementation in Java," Proc ICMC, September 1998.

[10] J. Piche, and A. Burton, ”Cecilia: A Production Interface to Csound”, Computer Music Journal Volume 22, Number 2, Summer 1998.

[11] http://www.saol.net

[12] J. Coutaz, ”PAC, an Implementation Model for Dialog Design”, Proc. of Interact’87, Stuttgart, September, 1987, pp. 431-436.

[13] http://www.oberlin.edu/ pblasser/rocky.html

[14] http://www.cs.berkeley.edu/ lazzaro/sa/index.html