Octopus Music API: Modelling Musical Performance

[1]Leandro Costalonga, Eduardo Miranda, [2]Evandro Miletto

[1]Interdisciplinary Centre of Computer Music Research - University of Plymouth
206 Smeaton Building,Drake Circus,Plymouth – UK

[2]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{leandro.costalonga,eduardo.miranda}@plymouth.ac.uk,
miletto@inf.ufrgs.br

Abstract. Music languages are on the cutting edge of all kinds of work in computer music [Loy and Abbott, 1985]. A satisfactory realization of an encoded work can be reconstituted through the interpretive practice of trained performers, but the knowledge that enables human performers to interpret music notation is extremely difficult to represent in a formal way [Sundberg et al., 1983]. Unlike previous work, the Octopus Music API aims to facilitate the programming of music performance system by providing the means to model not only the music but also the interaction of the performer and the musical instrument.

1. Introduction

A computer language presents an abstract model of computation that allows one to write a program without worrying about details that are not relevant to the problem domain of the program [McCartney, 2002]. Many are the ways in which computers can be employed in the service of music however the development of programming languages specifically for musical applications seems to have been concentrated on the areas of sound synthesis and musical performance [Loy and Abbott, 1985]. These languages are designed to provide a set of abstractions that makes expressing compositional and signal processing ideas as easy and direct as possible [McCartney, 2002].

The Octopus Music API is a Java API design to help in the modelling musical performance applications. Numerous software packages have been written for applications in music composition, music analysis, sound synthesis, and sound manipulation [Pennycook, 1985] but differently from other Java APIs' for music software development, the Octopus API has its focus on the modelling of musical performance elements - mainly the performer and his instrument. Since the limited space available for this paper, only the most important classes of the API will be presented. For a full understanding of the capabilities of the API please refer to the complete documentation at http://cmr.soc.plymouth.ac.uk/software/octopus/index.html.

The sections of this paper are related to the conceptual classification of the structures of the API. Starting with Musical Data Structures in Section 2, followed by Musical Data Interpreters and Instrument Classes. A brief conclusion is presented in Section 6.

2. Musical Data Structures

Musical Data Structures is a computational formalization of musical concepts that can be manipulated by the API. This classification applies for classes that represent musical concepts, such as *Note* or *RhythmPattern*.

2.1. Class octopus.Note

Most computer music notations define a musical note as the specification of an acoustic event. In the traditional music notation a *Note* specifies a human gesture toward an instrument [Loy and Abbott, 1985]. For us, the *Note* is the smallest audible element handled by the API.

2.2. Class octopus.Chord

A *Chord* is a set of *Notes* played together or arpeggiated. The *Notes* that compose a *Chord* are encapsulated in *ChordNote* objects which are the *Notes* with its contextual information, in other words, its role (interval) within the *Chord*.

3. Rhythmic Classes

3.1. Class octopus.Bar

Notes are sometimes connected by curved lines called slurs to show their grouping into phrases [Loy and Abbott, 1985]. A *Bar, in the context of this work,* is simply a rhythmic phrase. It's a collection of the smallest rhythmic structure designed in the API. The *Bar.RhythmEvent* can be a note or rest with values between 0 and 1 for duration, dynamic, and accentuation. *RhythmEvents* can be linked together through the *tie* attribute.

3.2. Class octopus.RhythmPattern

In the Octopus Music API the rhythmic line is defined independently from the *Melody* or *Harmony* facilitating the manipulation of the *Notes* and/or the rhythmic data.

The *RhythmPattern* represents a monophonic rhythmic line. Both the Melody and the Harmony embody *RhythmPattern, which is* composed of *Bars*, *Marks* and *Returning Points*. The B*ars* are inserted sequentially so the input order is relevant.

3.3. Class octopus.Arpeggio

An Arpeggio is a set of *RhythmPatterns* played simultaneously; it is used to spread the notes of the *Chord* along its duration (voicing). Often the *Arpeggio* information is omitted on the score and its use varies upon to the technique and expressivity of the *Performer*. When a *Musician* is requested to play a *Harmony* using a particular *Arpeggio*, it will adapt the *Arpeggio* to the *Harmony*, repeating or stretching its duration to match the duration specified for the *Chords*.

3.4. Class octopus.Melody

Melody is a set of *Notes* disposed in a sequence and played according to a certain *RhythmPattern*.

3.5. Class octopus.Harmony

Harmony is a set of *Chords* played according to a *RhythmPattern* and *Arpeggio*. If an *Arpeggio* is not assigned for a *Chord* then all *Notes* of the *Chord* will sound simultaneously and lasts for the respective duration assign for the *Chord*. The duration of each *Chord* is the same as the *RhythmEvent* associated with it. There is no information on the *Harmony* class defining how the *Chords* should be played (i.e. chord's shapes). This knowledge belongs to the *Musician*.

3.6. Class octopus.Music

MusicalComponents are the objects that can be placed in time and played as a single unit known as *Music*. These components can be either a *Melody or Harmony.*

4. Musical Data Interpreters

4.1. Class octopus.Musician

The *Musician* is basically an interpreter of the playable musical structures like: *Scale, Melody, Harmony, Music, RhythmPattern* and so on. He knows how to read and play these structures in the most basic possible level. No instrument restriction is considered in this computation.

4.2. Class octopus.instument.Performer

As a subclass of *Musician*, *Performers* also interpret musical structure but they have to adapt these *MusicalComponents* to the limitations of his *Instrument*. For instance, when a *Guitarist* plays a *Harmony* he will play it respecting the limitation of the *Guitar* he is playing which may sound slightly different when played by the simple *Musician*, although a *Guitarist* is a *Musician* in a higher level.

4.3. Class octopus.instument.fretted.Guitarist

The *Guitarist* is a *HarmonicPerformer* that knows how o play *Guitar* (actually, any fretted instrument). The *Guitar* used by the *Guitarist* has a direct influence in the way the *Music* is played.

As a *HarmonicPerformer*, the *Guitarist* needs to know how to play a *Chord* in the *Guitar*. A *Chord* can be played in several different ways (*ChordShape*) and using several different *Arpeggios*. Choose the most adequate *ChordShape* is a responsibility of the *Guitarist*. The *Guitarist* implemented in this API uses a similarity function to take this decision. Basically it chooses the most similar chord shape related to the previous one, minimizing the transitions effort [Costalonga et al. 2006].

5. Instrument Classes

5.1. Class octopus.instument.Instrument

The abstract class *Instrument* determines the minimum requirements that a new *Instrument* must implement to be able to interact with the other classes of API. This assures the scalability of the API to contemplate new *Instruments* during its development.

5.2. Class octopus.instument.string.fretted.FrettedInstrument

The *FrettedInstrument* Class represents the category of *Instruments* that have a string running along its fretted neck. The *Guitar* is an example of a subclass of *FrettedInstrument* that models a standard acoustic guitar, with 6 strings, 12 frets and standard tuning;

6. Final Words

The API is currently been hosted by SourceForge.net under Academic Free License. Free download and detailed information is available at http://cmr.soc.plymouth.ac.uk/software/octopus/.

This paper presented an overview of the Octopus Music API which is a Java API designed to assist software developers in the construction of software for musical performance. The API models what we believe to be the 3 elements presented in a musical performance: a) the performer, b) the instrument, and c) the music.

The key point to be observed in this API is the inverse approach of common music performance modelling, this means, instead of encode the performance task in complex musical notations we decided to endow the *Performer* with the ability and freedom to interpreter these structures based on his own knowledge. In the first stage, to prove the idea, the API focuses on a Guitar Musical Performance. Future work aims to extend the idea to different musical *Instruments*.

7. Acknowledgement

This research is sponsored by CAPES, Ministry of Education of Brazil.

References

Costalonga, L., Miranda, E., Matthias, J., Vicari, R. (2006) An Idiomatic Plucked String Player. In: Proc. of FLAIR's Special Track on Artificial Intelligence in Music and Art, Melbourne, Florida, USA.

Loy, G. and Abbott, C. (1985). Programming languages for computer music synthesis, performance, and composition. ACM Computing Surveys (CSUR), 17(2):235-265.

McCartney, J. (2002). Rethinking the Computer Music Language: SuperCollider. Computer Music Journal, 26(4):61-68.

Pennycook, B. (1985). Computer-music interfaces: a survey. ACM Computing Surveys (CSUR), 17(2):267-289.

Sundberg, J., Askenfelt, A., Fryden, L. (1983). Musical performance: A synthesis-by-ruleapproach. Comput. Music J. 7, 1.