IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

REVIEW ON DOMAIN SPECIFIC LANGUAGES FOR AUDIO PROGRAMMING

[1]Telidevulapalli Vijay Shankar

[1]IV B.Tech Student

[1] Department of Computer Science and Engineering

[1]ACE Engineering College, Hyderabad, Telangana, India.

Abstract: An audio programming language Specialized language intended production of computer music .
Unlike general-purpose programming languages audio programming languages have a notion of timing i.e. they are made to follow beats and pace by nature music programming languages follow data flow paradigm making audio programming languages advanced practice in language implementation.
Audio programming languages combine art of programming with music synthesis enabling algorithmic synthesis and various other composition styles otherwise hard to produce using traditional tools. There have been many audio programming languages in the past, each of them using different paradigms of programming and technique, existing languages, their paradigms and effectiveness are thoroughly discussed in this review article. It focuses on the level of granularity and effectiveness that the language model offers over audio production. Additionally, it covers the experimental findings related to each of the aforementioned language models.

Index Terms **- Audio programming, audio synthesis, Programming paradigm, music representation, algorithmic composition, live coding.**

I.INTRODUCTION

Though not a programming language traditional music notation features control flow in form of repetition[1] though granular control over sound produced leaves things to be desired it shows usage of notations to represent sounds over the years music has been improving and has become more sophisticated by day the development of computers only accelerated music production substantially special tools have been produced to create music without any special equipment and produce in one own computer without need for any special instrument as an example Kontakt7 has samples recorded from 900 different instruments over the time live coding and algorithmic composition have become one of the specialized ways to produce music hence there have been many programming languages each with incremental improvement and something new over the last one. Though traditional computation is sequential in music it is complicated as music is time sensitive, many things may happen at once parallelly and based on granularity data flows through many systems to produce sound as such traditional model of computation doesn't work as such a music programming language should enable things to happen parallelly and be time time sensitive and be data-flow in nature perhaps trying to make a music programming language will enable creation of not only good music production system but also good programming language. Before the discussion on different audio programming languages begins it's a good idea to discuss mechanisms by which these languages maintain properties of music as it is essential bread and butter to any system that involves production of music.

2. Fundamental parts of music programming Languages

Following section overviews parts of audio programming language that are most important to aspect of audio programming

2.1 Representing Time

At the end of day music is a combination of different forms of sounds that is pleasing to the ear. Sound being variations in air pressure over time make music exist in the time domain so it is important for programming languages dealing with music to have good time keeping capabilities. Below are some techniques discussed that are used by existing languages.

Non-preemptive threads

Using threads or coroutines is one technique to represent concurrency and exact logical timing.
Multiple calculations can run simultaneously instead of sequentially thanks to threads.
By adding "sleep" or "wait" routines that stop one thread's calculation while perhaps allowing other threads to continue, precise timing can be achieved.
Calling a "sleep" function in a traditional programme would cause an approximately timed pause, after which the thread would be scheduled to continue when the chance arose.
Of course, this might result in a buildup of minute timing inaccuracies, which could be detrimental in musical applications.

Many computer music languages employ the technique of keeping track of the logical time within each thread as a solution.
The logical time of the thread is advanced by a specific amount while it "sleeps."

Logical Time

One of the core things in computer simulations isd logical time i.e. how much time has the simulation run for. The same concept of maintaining a logical clock inside a program allows precise time computational events.
A real-time system can compute the next event earlier to catch up if the logical time is less than real time; however, if the logical time is greater than real time, the system can wait. As a result, systems that are based on precise logical times can avoid timing errors.

Beats and Tempo

Music systems frequently represent pace and beats in addition to logical time, which effectively "warps" or "deforms" musical time in relation to actual time. FORMULA (1990).

had complex mechanisms for tempo and temporal distortion in an early system. Temporal changes can be hierarchical in FORMULA and are precisely scheduled occurrences.
For instance, one process may control the pace while another, working within the established tempo parameters, might employ a brief tempo change known as a rubato.

2.2 Synthesis

Music is highly synchronous and there are many interconnected elements as such sound synthesis falls under data flow computation[2]. several methodologies have been used in programming languages to simulate the behavior of music some of them are briefed below.

Objects

One way to represent elements of nodes is via objects as in objects from object oriented programming and link objects together in this form of computation We do not see sounds as "values," and we do not rely on the language and runtime system to efficiently implement them. We view sound as an object in this approach once we link objects we may call compute on the root node and the root node will recursively call other nodes. In real-time systems, objects have the benefit of being able to be manipulated or updated to modify their behavior.

Graph

Languages like MaxMSP[3], Puredata[4] use visual paradigms to model graphs to represent data flow effect and process data.

Functional Audio Programming

Functional programming languages rely on lazy evaluation to do signal processing; they represent graphs as nested functions. Nested functions form a tree but with help of variable an acrylic graph can be formed but because of the way these languages are built the user is forced to use language inbuilt signal processing functions to compose signal this has restrictions which don't account for custom signal primitives

Imperative/Block structured computation

For real-time systems in particular, music audio calculation speed might be a serious issue. Compute sample data in vectors or blocks to increase computing efficiency. The software must call functions, follow connections to objects, and load coefficients into registers in order to compute audio. This overhead may consume as much time as the calculations performed on the samples.
Instead of computing sample-by-sample, block computation computes block-by-block. This allows for the amortization of a large portion of the computational cost across a number of samples. This may seem like a little detail, but it may speed things up by a factor of two.
Unfortunately, block computing poses a number of challenges to language designers that haven't yet been overcome.
Block computation's primary drawback is that logical computation durations are quantized in bigger increments than would be the case for individual samples, which generally have a period of 22 s. Blocks of 64 samples, which is a popular choice, have durations of 1.4 ms. In some cases of signal processing, 1.4 ms is simply not accurate enough. Here's a straightforward illustration: imagine that the linear segments that make up an amplitude envelope are capable of changing slope at each block boundary. As a result, the rising time of a sudden onset can only be 0 ms, 1.4 ms, or 2.8 ms. We can tell the difference in sound quality between these various rising periods.

3. BRIEF HISTORY OF AUDIO PROGRAMMING LANGUAGES

Acoustic programming languages are as old as early digital computers most of these languages can be traced back to the 90's below is a brief history of these audio programming languages.

3.1. MUSIC-N

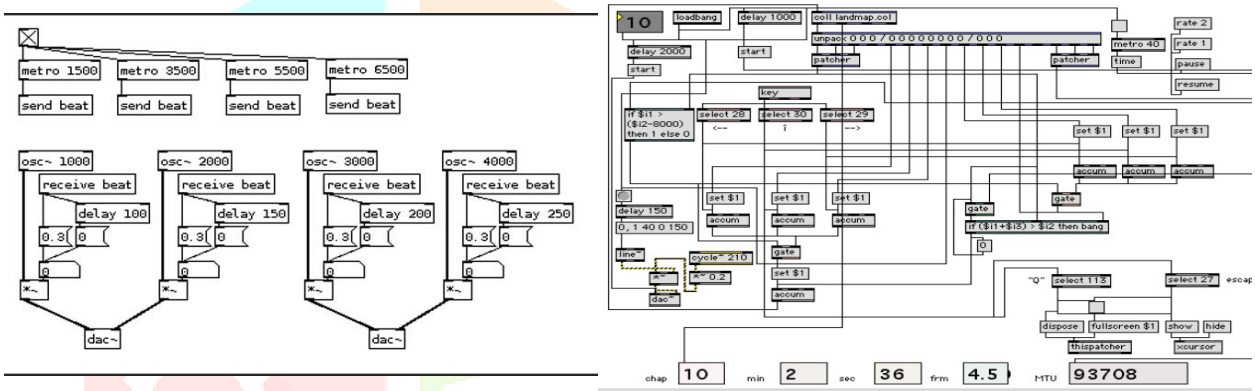
Music-1 or simply MUSIC is the earliest known music programming language The N in music-N refers to its descendents that took inspiration form original MUSIC written by Max Mathews . MUSIC was one of the earliest programmes for creating music on a digital computer, and it was undoubtedly the first software to achieve widespread recognition as a viable option in music research. MUSIC N languages let users specify modules interconnect them and specify timing to synthesize music. Based on this model MUSIC III gave rise to concept of Unit generators often called Ugen in short Ugens are like Library modules that come with MUSIC languages Ugen have several functions/generators often part of any digital audio workbench such as oscillators envelope's users of language interconnect these modules to create a patch it is to be noted that most MUSIC N languages until MUSIC 4 were implemented in PUNCH CARDS with MUSIC V being implemented in FORTRAN finally moving away from punch cards improving portability of MUSIC language.

3.2. AGE OF C LANGUAGES

As C languages became popular there have been many audio library with Cmix[5] taking a notable place
the most extensively used direct descendant of MUSIC-N is Csound, which was created in the late 1980s by Barry Vercoe and colleagues at MIT Media Labs.It recognises unit generators as opcodes, which are objects that create or manipulate audio.It adheres to the instrument vs. score paradigm, with instruments described in orchestra (.orc) files and scores defined in.sco files.Csound also supports the concept of independent audio and control rates.The audio rate (also known as the sample rate) is the rate at which audio samples are processed by the system.Control rate, on the other hand, governs how often control signals are computed and conveyed across the system.

3.3. GRAPHICAL LANGUAGES

Visual programming languages such as Max/msp and Pure data use a visual paradigm to represent audio processing units as nodes in a graph; users usually create music by customizing/modifying these nodes and linking them together.



(example Pure data and Max/msp patches)

4. Brief about notable languages

This section gives an overview of a few notable languages over the years note that this section omits languages mentioned in the history section.

4.1. Super collider

James McCartney first published SuperCollider[6] in 1996 for real-time music synthesis. It is a dynamically typed, single-inheritance, single-argument dispatch, garbage-collected, object-oriented language. Everything in SuperCollider is an object, including the most fundamental kinds like letters and integers. SuperCollider classifies its objects into different types. A unit generator is abstracted by the UGen class, while a set of unit generators working together to produce output is represented by the Synth class. An instrument is built with functionality in mind. To put it another way, when one constructs a function for sound processing, they are actually building a function for creating and connecting unit generators. A network of unit generator specification is not the same as a procedural or static object specification. Super Collider instrument functions can create a network of unit generators by utilizing the language's full algorithmic potential.

4.2. Chuck

ChucK[7] was created and chiefly designed by Ge Wang working with Perry R. Cook For real-time synthesis, composition, and performance, ChucK is a brand-new audio programming language that works with common operating systems. Concurrency, numerous, concurrent, dynamic control rates, and the ability to add, delete, and edit code on-the-fly while the application is running without pausing or restarting are all natively supported by ChucK.
It provides musicians and composers with a robust and adaptable programming tool for creating and experimenting with sophisticated audio synthesis programmes as well as real-time interactive control.

4.3. Faust

The GRAME-CNCM Research Department developed Faust[8] (Functional Audio Stream), a functional programming language for sound synthesis and audio processing with a special emphasis on the construction of synthesizers, musical instruments, audio effects, etc. For a range of platforms and standards, Faust focuses on high-performance signal processing programmes and audio plug-ins.

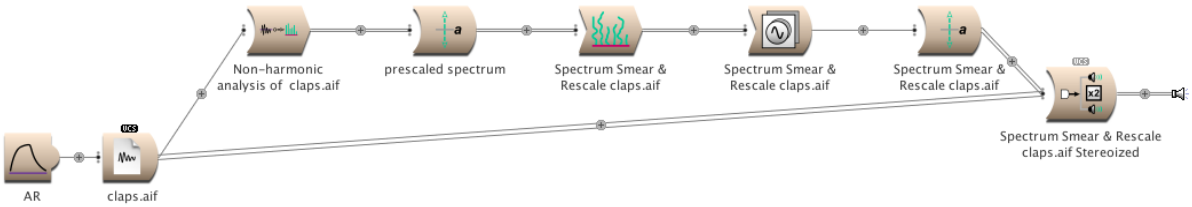
The compiler is Faust's primary element. Any Faust digital signal processing (DSP) specification may be "translated" into a variety of non-domain specific languages, including C++ and C.
The codes produced by Faust are simple to compile into a broad range of objects, including audio plug-ins, standalone programmes, smartphone and web apps, etc.

4.4. Sonic Pi

Sonic-Pi[9] is a music live coding language that has been designed for educational use as a first programming language by Sam Aaron in the University of Cambridge Computer Laboratory in collaboration with Raspberry Pi Foundation.

4.5. Kyma

Kyma is a visual programming language used by sound designers, academics, and artists. By visually linking modules on the screen, a user programmes a multiprocessor DSP in Kyma. Carla Scaletti, created the original version of Kyma in 1986 using the Smalltalk programming language to calculate digital audio samples on a Macintosh 512K. Kyma has characteristics of both object-oriented and functional programming languages. The basic unit in Kyma is the "Sound" object upon which unary n-ary transformations are made.



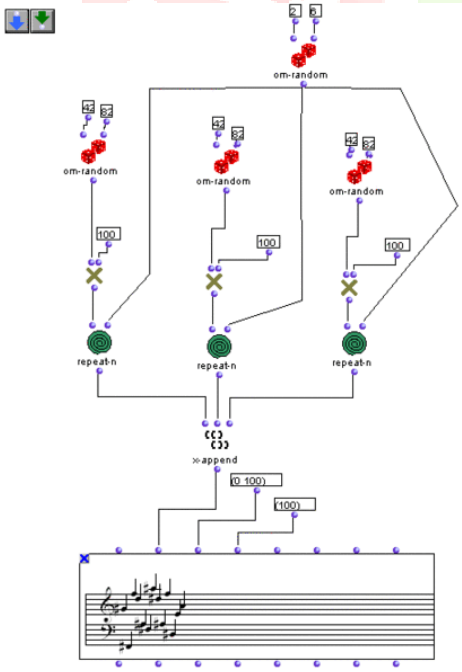
(example kyma patch)

4.6. GLICOL

GLICOL is designed by Qichao Lan and Alexander RefsumGLICOL stands for graph-oriented live coding language . This language is designed to represent directed acyclic graphs(DAG), using a syntax optimized for live music performances.GLICOL is designed in RUST programming language its mainly intended for live coding as it supports co-performance with the support for collaborative editing.

4.6. OpenMusic

OpenMusic (OM) is a Common Lisp-based visual programming language. Visual programmes are built by linking and constructing icons that represent functions and data structures. The majority of programming and operations are carried out by dragging an icon from one location and placing it in another. There are built-in visual control structures (such as loops) that interact with Lisp ones. Existing CommonLisp/CLOS code can be readily ported to OM, and new code may be created visually.

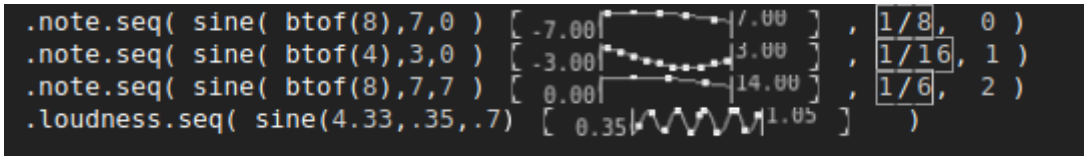


(path created by SecretTheatre submitted to wiki)

5. Future prospects

There are many audio programming languages as time goes on hardware improves by day and it opens new doors for newer technologies,language development is ever evolving there are many languages that are being developed that are inherent to their predecessors for example faust solves issue of unify

the score and orchestra one can create a hybrid visual system to facilitate understanding of real time music systems for users of language. Gibber[10] is one such language; it animates sources to indicate what is happening.



(Visualization made on running Gibber program on web)

Conclusion

The potential to language design and language architecture that audio programming is immense with the rise of live coding and algorithmic composition audio programming will prevail. Newer languages only make audio programming more accessible to everyone.

REFERENCES

[1] Jin, Zeyu and Roger B. Dannenberg. "Formal Semantics for Music Notation control Flow." International Conference on Mathematics and Computing (2013).

[2] Bernardini, Nicola & Rocchesso, Davide. (1970). Making Sounds with Numbers: A Tutorial on Music Software Dedicated to Digital Audio. Journal of New Music Research. 31. 10.1076/jnmr.31.2.141.8089.

[3] Bevilacqua, Frédéric & Müller, Remy & Schnell, Norbert. (2005). MnM: a Max/MSP mapping toolbox. 85-88.

[4] Puckette, Miller. (1970). Pure Data: another integrated computer music environment.

[5] Pope, Stephen. (1996). Machine Tongues XV: Three Packages for Software Sound Synthesis. Computer Music Journal. 17. 10.2307/3680868.

[6] McCartney, James. (2002). Rethinking the Computer Music Language: SuperCollider. Computer Music Journal - COMPUT MUSIC J. 26. 61-68. 10.1162/014892602320991383.

[6] Wang, Ge & Cook, Perry & Misra, Ananya. (2005). Designing and Implementing the ChucK Programming Language.

[7] Orlarey, Yann, Dominique Fober and Stéphane Letz. "FAUST : an Efficient Functional Approach to DSP Programming." (2009).

[8] Aaron, Sam. (2016). Sonic Pi – performance in education, technology and art. International Journal of Performance Arts and Digital Media. 12. 171-178. 10.1080/14794713.2016.1227593.

[9] Scaletti, Carla. (2002). Computer Music Languages, Kyma, and the Future. Computer Music Journal. 26. 69-82. 10.1162/014892602320991392.

[10] Roberts, Charlie & Kuchera-Morin, JoAnn. (2012). Gibber: Live coding audio in the browser. 64-69.