FESTIVAL INTERNACIONAL DE LA IMAGEN
15 FESTIVAL INTERNACIONAL DE LA IMAGEN
9-13 mayo / may 2016
Diseño + Arte + Ciencia + Tecnología
Manizales - Colombia
www.festivaldelaimagen.com

Diseño e implementación de un DSL : CQenze, como lenguaje de primera experiencia para el código en vivo.

Esteban Betancur
Licenciado en música, Profesional certificate in guitar de Berkley college of music, y actualmente aspirante a MDA del ITM
ITM - Maestría en artes digitales
Medellin, Colombia
estebanbetancur238346@correo.itm.edu.co

Abstract

La idea general es discutir los aspectos que se tuvieron en cuenta en el proceso de diseño e implementación de la sintaxis empleada en el lenguaje de dominio específico (DSL) CQenze, que es un lenguaje orientado al livecoding con una aproximación que se enfoca en la visualización o mejor dicho, la abstracción del código desde una perspectiva mas "gráfica" permitiendo que el programador visualice la concurrencia de los eventos de manera clara.

Buscaremos entender el porqué es necesario diseñar nuevas formas de sintaxis, alejadas del paradigma imperante dada la complejidad de la implementación temporal de los eventos gráficos y sobre todo los musicales, por esta razón se expondrán varias perspectivas y diferentes enfoques sintácticos de otros lenguajes que han intentado solucionar el problema del tiempo y la concurrencia.

Se expondrán las razones que hacen de CQenze un lenguaje ideal para la primera experiencia con código en vivo y sus posibilidades pedagógicas para el trabajo con EUP (end-user programmers) desde niños, con el único requerimiento de que sean capaces de leer y escribir en el teclado, ademas de las cualidades (fácil implementación, compilador ligero, alto desempeño atado a la posibilidad de real-time que le otorga su motor chucK) y limitaciones que este tiene al ser diseñado para alcanzar objetivos "textiles" y "gráficos" tan específicos.

Por último se plantearán las posibles expansiones que este lenguaje debería implementar motivadas por el uso y la experiencia.

Palabras clave

Sonología, Composición algorítmica, DSL (lenguajes de programación de dominio especifico), sintaxis, livecoding, computación musical, chucK.

Mesa temática

Sonología.

Categoría

Ponencia.

Introducción

Con la aparición del live coding y los lenguajes de programación especializados o de dominio especifico, empezaron a surgir necesidades propias de estos programadores que a diferencia de la perspectiva tradicional, donde el programador no siempre necesita "ver" (en nuestro caso diremos escuchar) el resultado de su código, en el live coding si lo necesita por la "naturaleza" del término live (en vivo); es decir, es requerido que los resultados de cada cambio en el código sean percibidos al instante; el live coding presenta retos computacionales nuevos, lo que llevó a la creación de respuestas originales; por ejemplo el problema de agendar varios subprocesos (threads scheduling), dado que la música requiere la ejecución simultánea de diferentes procesos y el garantizar que estos subprocesos se ejecuten en el momento que se requiera (real-time) desembocó en el desarrollo de chucK (Wang, 2004) y a su heredero en este concepto Sonic PI (Aaron, Orchard, Blackwell, 2014).

El problema "en vivo"

El problema de la sintaxis, dado que los lenguajes de programación no nacieron para adecuarse a las abstracciones musicales, digamos ritmo, melodías, armonía, tienden a hacer que el programador pase mucho tiempo escribiendo código antes de escuchar los primeros eventos, en palabras de Magnusson:

> "There is little fun in watching a stressed programmer designing algorithms for minutes before a simple sine oscillator is applied in the playback of a silly melody". [1]

A este respecto, podemos mencionar ixi-lang (Magnusson, 2010) y Tidal (McLean, 2014) como mini lenguajes diseñados pensando en adecuarse mejor a dichas abstracciones. En el primer caso, ixi-lang usando SuperCollider como motor y compilador, Tidal usando Haskell como lenguaje base pero actualmente migrando su motor de audio de C a SuperCollider.

La sintaxis musical

Los lenguajes de sintaxis tradicional permiten una infinita cantidad de posibles algoritmos para la creación de música, en el caso de chucK la sintaxis es bastante cercana a C++, Sonic PI, por otro lado está en el estilo de Ruby, el problema surge con el nacimiento del movimiento Algorave (Collins, McLean, 2014) y la necesidad de mantener un público bailando, lo que le suma otras dificultades al live coder además de las ya mencionadas; la más importante es que su música debe ser rítmica pero al mismo tiempo debe tener la posibilidad de ser cambiada rápidamente, y es ahi donde falla la sintaxis que no ha sido especialmente diseñada para la adaptación a las abstracciones musicales.

ChucK **Tidal** **CQenze**

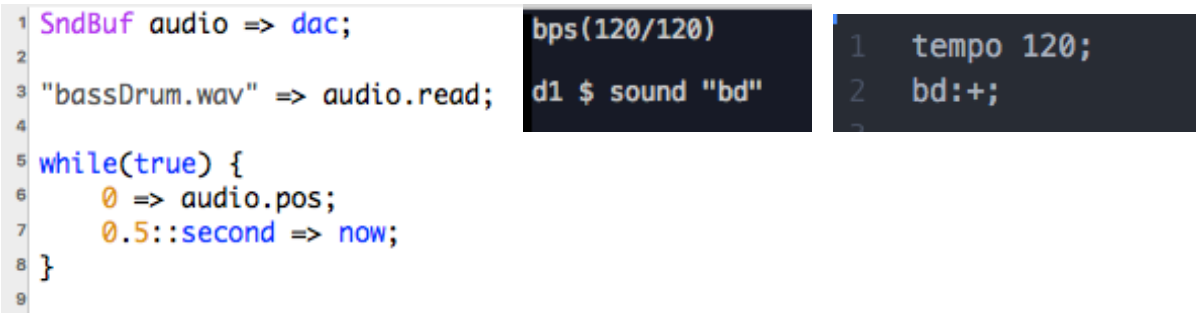

Figura 1. Ejemplo de la sintaxis necesaria para lanzar un sample en chucK de manera rítmica comparada con el mismo patrón en Tidal y en CQenze. La asignación del tempo en este ejemplo no seria necesaria dado que se usa 120 que es el valor por defecto (propósitos ilustrativos).

Como se puede observar en la ilustración, la sintaxis de chucK esta a mas bajo nivel, lo que la hace ideal en situaciones diacrónicas, es decir, donde la producción del sonido no debe ser simultánea al hecho musical y en el caso del club, al baile; ahí es justamente donde radica su debilidad para la ejecución en vivo.

Para resolver este problema hay dos enfoques: el primero es desarrollar un entorno interno, basado en el mismo lenguaje que permita elevar el nivel; en el caso de chuck, dado que es un lenguaje orientado a objetos, un conjunto de clases que engloben cada una de las abstracciones bajo el modelo de herencia de atributos y con esta ayuda, lanzar a mas alto nivel las ideas del programador, la segunda, es usar la potencia del lenguaje en si mismo y su motor de síntesis pero no su modelo sintáctico y superponerle una capa sintáctica diseñada específicamente para las abstracciones musicales, en el caso de CQenze, el ritmo.

El modelo sintáctico: Secuenciador de pasos o matrices

El modelo sintáctico de CQenze esta basado en el concepto de matriz, donde cada posición del vector determina el estado (activado, desactivado) de un evento, es decir, el vector funciona como un secuenciador de pasos o drum machine haciendo una referencia directa a la abstracción que se pretende representar.

∫Cada vector esta determinado por un objeto pre instanciado, es decir, al poner una carpeta con samples en la subcarpeta de dirt (McLean, 2014) en CQenze, el compilador la instancia como un objeto válido del lenguaje, luego es posible escoger los samples dentro del objeto/carpeta con "/" así:

"bd/2" , del objeto/carpeta "bd" cargar un buffer con el sample número dos de la carpeta.

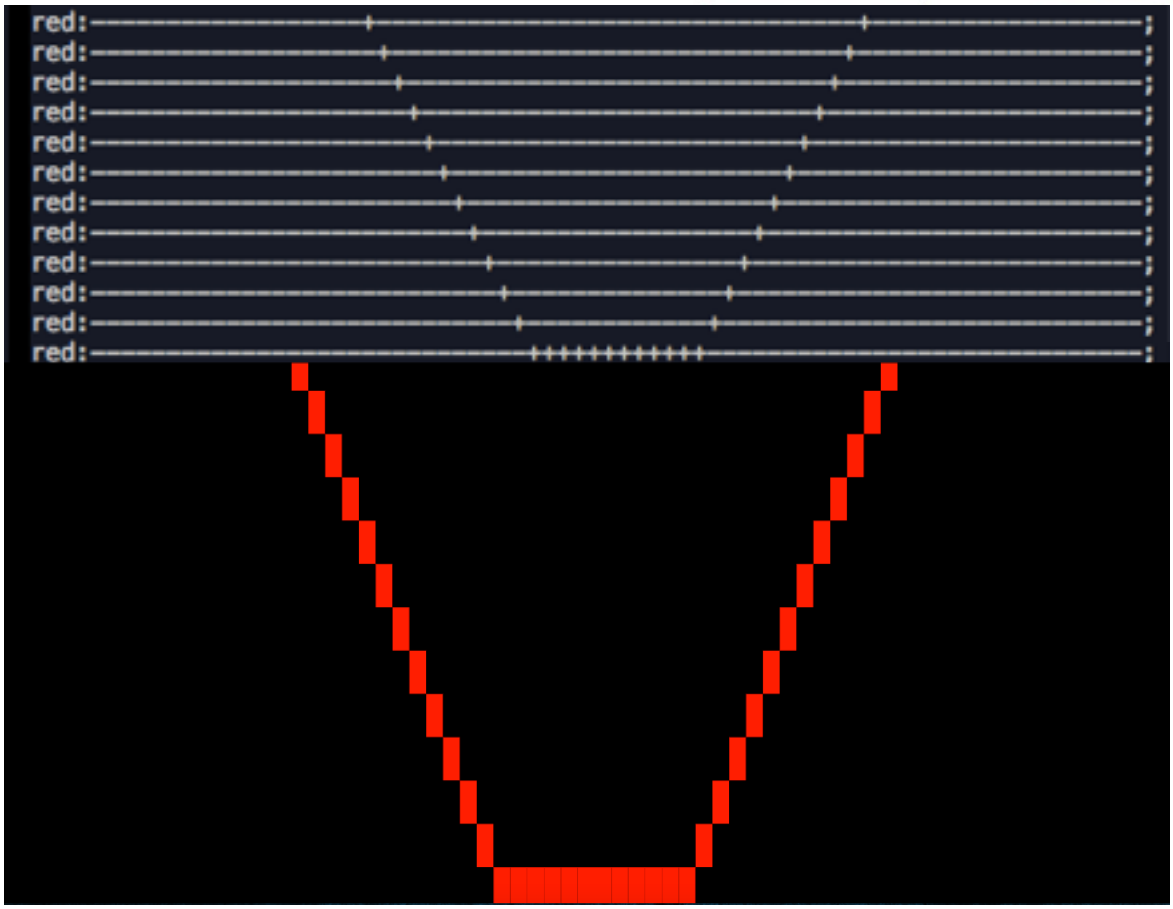

Figura 2. Código y visualización de CQenze usando javascript y html5.

Luego el operador de asignación, que puede ser "=" ó ":"; cada uno con una funcionalidad temporal distinta.

El operador ":" distribuye los eventos forzando los samples a ser reproducidos en eventos simétricos durante la cantidad de ciclos determinados (por defecto dos), es decir, cada evento programado tendrá la misma cantidad de tiempo que los demás en el mismo vector, y este tiempo de ejecución será relativo a la cantidad de eventos . El otro operador "=" corta los samples a negras (quarter notes, un futuro desarrollo es el uso de diferentes subdivisiones) y fue implementado con la intención de usar notación convencional.

Ahora, los eventos se programan en un vector compuesto por "+" y "-" lo que indica activado/ desactivado cerrando por un punto y coma ";".

De esta forma, se presenta como un mini lenguaje mas "gráfico" en el sentido que es posible para el

programador visualizar la concurrencia de los eventos con solo ver el código.

Es por esta razón por la que CQenze se convierte en la herramienta ideal para demostrar la relación texto-proceso que se produce en la programación en especial para personas que no tienen ninguna experiencia (EUP) ya que al producir un resultado instantáneo y evidente (Aaron, Orchard, Hoadley, Regan 2011) no necesita de capacitación previa y tiene una alta capacidad de impacto sobretodo en la primera experiencia; claramente es esto mismo lo que puede ser su debilidad a largo plazo, su simplicidad.

Implementación

CQenze es una capa sintáctica sobre chuck, sobre el cual actúa como "parásito", usando el motor de síntesis, y las capacidades específicas que lo hacen tan potente: tiempo real (real time), y su famoso "spork~" (Ejecución paralela). Experimentalmente, puede servir como capa para javascript/html5 y generar patrones de color.

La construcción del lenguaje tiene varias capas interconectadas, siendo la primera su analizador léxico escrito para Lex/Flex, este "scanner" recibe el input del usuario que es escrito en un archivo llamado Live.cqz ubicado en la misma carpeta y lo envía a el interprete directamente, que hace las veces de "parser generator" y de "transpilador" hacia chucK (javascript/html5 en el caso gráfico). Este interprete está escrito en python y es el encargado de generar los mensajes de compilación, activar la maquina virtual de chucK y enviar el código para su ejecución.

CQenze acepta comentarios de línea usando la sintaxis tipo C "//", y frases válidas que deben tener la estructura objeto + operador + vector + ";", en caso de no ser válida, el interprete genera unos diez mensajes de error diferentes según sea el caso.

CQenze está implementado sobretodo como lenguaje para una primera experiencia en programación por lo que tiene una funcionalidad basada en Extramuros (Ogborn, Tsabary, Jarvis, Cardenas, McLean, 2015), que es un entorno para compartir buffers de texto, para que no requiera instalación y pueda ser probado con un servidor externo y el código escrito y lanzado desde el explorador web usando cualquier dispositivo con acceso a internet o a una red local. En ese caso sólo el servidor debe tener chucK y el interprete de CQenze.

Conclusion y trabajo futuro

El concepto de pasos o el de vector usado simplemente como drum machine demuestra gran potencia en lo que respecta a la posibilidad de visualización de las abstracciones musicales, en especial del ritmo pero al mismo tiempo limita las posibilidades a largo plazo, aunque este concepto puede ser extendido a la melodía y la armonía dentro de un futuro desarrollo del lenguaje ampliando su uso a la posibilidad de determinar eventos específicos rítmicamente, como uso de filtros y efectos, generadores melódicos de pasos o la implementación complementaria de librerías como CHmUsiCK (https://github.com/essteban/CHmUsiCK) en la sintaxis de CQenze con lo que se puede extender el lenguaje al uso de funciones con sintaxis punto (.).

Referencias

[1] Magnusson, T. (2010). *ixi lang: A SuperCollider parasite for live coding*. [SuperCollider Symposium 2010] en http://www.ixi-software.net/content/backyard.html

Bibliografía

[1] Aaron, S., Orchard, D. & Blackwell, A. (2014). *Temporal semantics for a live coding language.* En Proc. Farm' 14.
[2] Aaron, S., Blackwell, A., Hoadley, R. & Regan, T. (2011) *A principled approach to developing new languages for live coding.* En Proc. NIME
[3] Collins, N. and McLean, A. (2014). *Algorave: Live Performance of Algorithmic Electronic Dance Music*. En Proc. NIME.
[4] Magnusson, T. (2010). *ixi lang: A SuperCollider parasite for live coding*. [SuperCollider Symposium 2010] en http://www.ixi-software.net/content/backyard.html
[5] McLean, A. (2014). *Making programming languages to dance to: Live Coding with Tidal.* En Proc. Farm' 14.
[6] Ogborn, D., Tsabary, E., Jarvis, I., Cardenas, A. & McLean, A. (2015) *Extramuros: making music in a browser-based, language-neutral collaborative live coding environment.* En Proc. ICLC
[7] Wang, G., & Cook. P. R. (2004). *On-the-fly programming: Using code as an expressive musical instrument*. En Proc. NIME.

Biografía de el Autor

Licenciado en música, con un profesional certificate in guitar de Berkley college of music, y actualmente aspirante a MDA del ITM en la ciudad de Medellin, a trabajado como investigador del laboratorio del programa Medellin vive la música en el desarrollo herramientas para la creación desde el código en vivo.
Director y fundador del programa de teatro musical del Pequeño teatro.

http://scienti.colciencias.gov.co:8081/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0000045419

Reconocimientos

Un reconocimiento especial al grupo Algo~ritmos como espacio de experimentación, y sobre todo a Federico Lopez por su guía en la investigación.