ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

Análisis Comparativo de Plataformas para el Live Coding

Molina Quintero, Diego[1(*)]; Montes Restrepo, Victoria[1]

[1]Instituto Tecnológico Metropolitano, Facultad de Artes y Humanidades, Medellín, Colombia.

RESUMEN

En este trabajo se realiza una comparación entre tres plataformas para *live coding*, es decir, la creación en tiempo real de sonidos y ritmos por medio de expresiones matemáticas. Las orquestas o ensambles de computadores son las agrupaciones que usan el *live coding* como medio para interpretar la música. En estos ensambles, cada ordenador debe sincronizarse con el servidor, considerando que múltiples instrumentos se tocan simultáneamente. Así, el hallazgo de herramientas que sean versátiles a la hora de interconectar computadores y que se ajuste al tipo de usuario, ya sea músico o informático, es uno de los desafíos centrales en el *live coding.* Dado lo anterior, se busca comparar la funcionalidad y practicidad de las herramientas actuales para la interpretación de música en una orquesta de computadoras. Por esta razón, se seleccionaron tres plataformas usadas en los eventos de creación de música por computadora: Estuary, Gibber y Troop. Dichas plataformas se compararon con respecto a la facilidad de instalación, cantidad de DSL (lenguajes de dominio específico) soportados, tiempo de conexión, y estadísticas en GitHub. Como resultado del análisis comparativo, se encontró que cada plataforma se puede ajustar a un tipo de usuario diferente, generando las siguientes recomendaciones: (i) Estuary para personas que inician en el *live coding*; (ii) Gibber para presentaciones rápidas, ensambles esporádicos y usuarios con conocimientos básicos de *live coding*; (iii) Troop para usuarios con conocimientos avanzados en el *live coding.* Adicionalmente, se propone la implementación de la plataforma SoOrOk, como alternativa que integrará utilidades de las plataformas analizadas, permitiendo su uso por parte de personas con conocimientos básicos o avanzados de *live coding.*

Palabras clave: Orquesta de computadoras. Estuary. Troop. Gibber. Live coding. DSL. SoOrok.

Comparative Analysis of Platforms for Live Coding

ABSTRACT

In this work, we perform a comparison between three platforms for live coding, i.e., the real-time creation of sounds and rhythms through mathematical expressions. The orchestras or computer ensembles are groups that use live coding as a means to interpret music. In these ensembles, each computer must synchronize with the server, considering that multiple instruments are played simultaneously. Thus, finding tools that are versatile in relation to interconnecting computers, and that fit the type of user, be it a musician or a computer scientist, is one of the central challenges in live coding. Given the above, we seek to compare the functionality and practicality of current tools for music interpretation in a computer orchestra. For this reason, three platforms used in computer music creation events were selected: Estuary, Gibber, and Troop. These platforms were compared with respect to ease of installation, amount of supported DSL (domain-specific languages), connection time, and statistics in GitHub. As a result of the comparative analysis, we found that each platform can be adjusted to a different type of user, generating the following recommendations: (i) Estuary for beginners in live coding; (ii) Gibber for quick presentations, occasional assemblies, and users with basic live coding skills; (iii) Troop for users with advanced knowledge of live coding. In addition, the implementation of the SoOrOk platform is proposed, as an alternative that will integrate utilities from the analyzed platforms, allowing its use by people with basic or advanced live coding knowledge.

Keywords: Laptop orchestra. Estuary. Troop. Gibber. Live coding. DSL. SoOrok.

Recibido: 30/11/2021 Aceptado: 17/12/2021
Correspondencia: (*) diegomolina97685@correo.itm.edu.co

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

1. INTRODUCCIÓN

Los algoritmos se definen como un conjunto de instrucciones que, combinadas apropiadamente, permiten escribir programas que son entendibles y ejecutables por un computador (Trejos y Muñoz, 2020). La música como proceso creativo es algorítmica, se basa en reglas, métodos y sistemas que permiten la producción de material sonoro, el cual después es llamado obra o pieza musical. Estas reglas musicales han variado según el contexto social, político, económico y geográfico, es decir, los algoritmos se han transformado según la época y el lugar, creando así disciplinas, normas y fórmulas para la invención de música; es allí, donde aparecen los manuales de armonía, contrapunto, y la teoría musical y se instauran instituciones académicas en torno a sus propios algoritmos o formas de hacer música.

Desde hace algunos años, las computadoras, los celulares, las tabletas y otros dispositivos electrónicos empezaron a hacer parte importante de la creación e interpretación musical. El *live coding* surge como una práctica musical derivada de la programación, la cual permite la creación en tiempo real de sonidos y patrones con expresiones matemáticas. Dichas formas de interpretación permiten el desarrollo de música electrónica de carácter bailable, es decir, su uso actual está enfocado en la interpretación de música para discotecas *(raves*), creando así, festivales, concursos, convocatorias y becas en torno a ello (Molina et al., 2019).

Las orquestas de computadoras proponen una contradicción, una paradoja entre orquesta y computadora ya que la asociación de estos términos parece algo imposible si se tiene en cuenta que la orquesta es una institución casi primitiva, arcaica o anticuada y la palabra computadora hace referencia a una herramienta actual (Trueman, 2007).

Las orquestas o ensambles de computadores tienen un problema en común a la hora de sincronizar (musical y tecnológicamente) cada instrumento. Cada ordenador debe diseñar sus propios sonido-timbre (instrumento) y sincronizarse con anterioridad al servidor teniendo en cuenta que se tocan simultáneamente muchos instrumentos. Este es quizás el desafío central para los intérpretes musicales, ya que si el ensamble crece o cambia de género musical, es necesario crear nuevos instrumentos y esto implica una nueva sincronización con el servidor. Así, el hallazgo de herramientas que faciliten la interconexión rápida entre computadores y la creación sonora ágil, sin importar el tipo de usuario, se convierte en una necesidad para el uso de *live coding.*

Dado lo anterior, es importante realizar una valoración de las funcionalidades de las herramientas para *live coding* existentes actualmente. Por lo tanto, en este artículo se presenta una comparación entre diferentes plataformas usadas para *live coding:* Estuary, Gibber y Troop; se comparan desde el punto de vista de su instalación, un caso de uso, su tiempo de conexión y las estadísticas en GitHub. Adicionalmente, se propone la implementación de la plataforma SoOrOk, como alternativa que integra herramientas para músicos e informáticos.

2. MARCO TEÓRICO

Las *laptop orchestras* aparecen a finales del siglo XX y es a partir de ese momento que comienza a gestarse un nuevo paradigma musical donde el hecho de usar computadoras y manipular sonido para producir música se convierte en una actividad cada vez más cercana a la práctica de música tradicional.

Entre 1977 y 1983 aparece en San Francisco, E.E.U.U., la LAMC (*League of Automatic Music Composers*), considerada como el primer ensamble con características orquestales que incorporó computadoras en su hacer musical. La LAMC creó redes de computadoras interactivas y otros circuitos electrónicos con miras a provocar nuevas inteligencias musicales (Collins et al., 2003). Ellos lograron crear una agrupación compuesta de máquinas de música automáticas programadas de forma independiente, produciendo una música ruidosa, a menudo impredecible, y ocasionalmente hermosa. Para la música, esto significó redefinir

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

todo su hacer, desde los instrumentos y los sistemas de afinación hasta las formas musicales, lugares y relaciones sociales entre los músicos y el público.

Desde sus inicios, esta nueva aproximación al quehacer musical se ha visto altamente apoyada por grupos y/o personas interesadas en mejorar y ampliar las posibilidades del conocer, hacer y aprender dentro del campo, lo cual no sólo ha permitido una tecnificación de saberes, sino una apertura a la divulgación de algo que pudo haber quedado como un mero fenómeno aislado; es así que organizaciones como TOPLAP (*Temporary Organisation for The Promotion of Live Algorithm Programming)* abren un espacio de discusión y promoción de ensambles electrónicos (https://toplap.org/).

En los últimos años, el uso de herramientas para la creación colaborativa alrededor del *live coding* aumentó, apareciendo en los diferentes espacios, congresos y festivales herramientas con lenguajes *multiparadigma* y *multiplataforma.* Multiparadigma significa que con este lenguaje se pueden construir programas con enfoques diferentes, y multiplataforma que se puede utilizar en cualquier sistema operativo (Trejos y Muñoz, 2020).

Los DSL (*Domain Specific Language*) o lenguajes de dominio específico usados en live coding, se diferencian de los lenguajes de programación convencionales en la velocidad de su escritura. A diferencia de la sintaxis tradicional que permite una infinita cantidad de posibles algoritmos, los DSL tienen la posibilidad de ser cambiados rápidamente, y es ahí donde falla la sintaxis que no ha sido especialmente diseñada para la adaptación a las abstracciones musicales (Betancur, 2016). La Tabla 1 presenta un resumen de los principales DSL empleados en *live coding.*

Tabla 1. Principales DSL usados en *live coding.*

Lenguaje	Sistema operativo	GitHub	Referencia
TidalCycles (MiniTidal)	Microsoft Windows, Mac OS, Linux.	https://github.com/tidalcycles/Tidal	(McLean, 2014)
Super Collider	Microsoft Windows, Mac OS, Linux.	https://github.com/supercollider	(Wilson et al., 2011)
Sonic Pi	Microsoft Windows, Mac OS, Raspberry Pi OS.	https://github.com/sonic-pi-net/sonic-pi	(Aaron y Blackwell, 2013)
Python (FoxDot)	Microsoft Windows, Mac OS, Linux.	https://github.com/Qirky/FoxDot	(Kirkbride, 2016)
Punctual	Microsoft Windows, Mac OS, Linux.	https://github.com/dktr0/Punctual	(Littler et al., 2021)
CineVivo (CineCer0)	Microsoft Windows, Mac OS, Linux.	https://github.com/essteban/CineVivo	(Rodríguez et al., 2019)
glsl-fragment	Microsoft Windows, Mac OS, Linux.	https://github.com/rezaali/Fragment	(McKinney, 2014)
glsl-vertex	Microsoft Windows, Mac OS, Linux.	https://github.com/freeman-lab/glsl-basic-vertex-shader	
Hydra	Microsoft Windows, Mac OS, Linux.	https://github.com/ojack/hydra	(Sedláková, 2021)
CQenze	Microsoft Windows, Mac OS, Linux.	https://github.com/essteban/CQenze	(Betancur, 2016)

Algunos ejemplos de plataformas para *live coding*, que incorporan los lenguajes DSL presentados en la Tabla 1, son:

- **Estuary:** Es una plataforma que permite la colaboración y el aprendizaje a través del live coding. Facilita experimentar con sonido, música e imágenes en un navegador web. Además, es un software gratuito y de código abierto, publicado bajo los términos de la Licencia Pública GNU (versión 3).
- **Gibber:** Es una plataforma de live coding para el navegador web, que combina la síntesis y secuenciadores gráficos.
- **Troop:** Es una plataforma de colaboración alrededor del live coding que permite la codificación (escritura de código) grupal en vivo dentro del mismo documento en múltiples computadoras.

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

3. METODOLOGÍA

En esta sección se presenta la instalación de las tres plataformas de *live coding* colaborativo estudiadas: Estuary, Gibber y Troop. El equipo utilizado para la instalación y pruebas cuenta con las siguientes especificaciones: Sistema operativo Linux Ubuntu 20.04, Procesador Intel Core i5 de octava generación, 8 GB RAM, 500 GB SSD, Tarjeta de audio Focusrite 2i4. Adicionalmente, se realiza una revisión de las principales características de cada plataforma.

3.1. Instalación.

Estuary.

La instalación de Estuary requiere el uso de un gestor de paquetes. En este caso se utiliza `Nix` pues ofrece mejoras respecto a `apt`, `yum` y `dnf` y se acerca mucho a `Docker` y otros sistemas modernos (Devopedia, 2020).

Para instalar `Nix` se escribe en la terminal de Ubuntu:

```
$ sudo mkdir /etc/nix; echo 'use-sqlite-wal = false' | sudo tee -a /etc/nix/nix.conf
```

Se instala `git` en Debian/Ubuntu para poder clonar el repositorio.

```
$ sudo apt install git-all
```

Se clona el repositorio en la carpeta deseada.

```
$ git clone https://github.com/dktr0/Estuary
```

En la carpeta del proyecto se ejecuta `nix-build` para compilar todos los paquetes y crear los link simbólicos a la carpeta result.

Se crea el paquete de instalación:

```
$ make bundleClient
```

Esto generará un archivo llamado `estuary-client.zip` con la versión de producción del cliente, las dependencias de front-end y los activos estáticos (excluyendo las muestras de audio).

Se crea y se lanza una implementación local completa del servidor Estuary

```
$ make fullBuild
```

El proceso de compilación tardará algunos minutos. Una vez que se complete correctamente, habrá una implementación completa de Estuary en la carpeta de staging. Para generar certificados SSL temporales se usa:

```
$ make selfCertificates
```

Se inicia el servidor para verificar el procedimiento:

```
$ cd staging
$ ./EstuaryServer password 8000
```

Este servidor está escuchando en el puerto 8000. Se inicia en el navegador web (localmente), escribiendo https://127.0.0.1:8000.

Si se quiere usar un puerto específico se deben tener privilegios `root`:

```
$ cd staging
$ sudo ./EstuaryServer password 443
```

Con lo anterior se conecta al navegador web (localmente) con https://127.0.0.1:443.

Gibber.

Para usar Gibber se debe instalar `node/npm` en el equipo:

```
$ apt-get install -y nodejs
$ sudo apt-get install -y npm
```

Posteriormente, se clona el proyecto:

```
$ git clone https://github.com/gibber-cc/gibber.git
```

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

Ingresando a la carpeta del proyecto, se ejecuta:

```
$ npm i
$ npx gulp
```

Al terminar la instalación se ejecuta `npm start`, lo cual creará un servidor en el puerto 8080. Para ver Gibber en el navegador, se copia la dirección https://127.0.0.1:8080.

Troop.

En el caso de Troop es necesario tener Python 3 instalado.

```
$ sudo apt install python3

$ python --version
```

Para terminar la configuración de Python se instala su interfaz gráfica:

```
$ sudo apt-get install python-tk
```

Posteriormente, se debe instalar SuperCollider, un lenguaje para síntesis de audio y composición algorítmica, utilizado por músicos, artistas e investigadores que trabajan con sonido (ver Tabla 1):

```
$ sudo apt-get install build-essential cmake libjack-jackd2-dev libsndfile1-dev libfftw3-dev libxt-dev libavahi-client-dev

$ sudo apt-get install git libasound2-dev libicu-dev libreadline6-dev libudev-dev pkg-config libncurses5-dev

$ sudo apt-add-repository ppa:beineri/opt-qt-5.11.0-xenial

$ sudo apt-get update

$ sudo apt-get install qt511base qt511location qt511tools qt511webchannel qt511xmlpatterns qt511svg qt511webengine qt511websockets

$ sudo apt-add-repository ppa:beineri/opt-qt-5.11.0-xenial

$ sudo apt-get update

$ sudo apt-get install -y supercollider
```

Es importante instalar los plugins de SuperCollider ya que permite a Troop tener más opciones de uso. Se descarga y se descomprime este archivo en cualquier carpeta: https://github.com/supercollider/sc3-plugins/releases.

Para finalizar la configuración, se abre SuperCollider y se ejecutan las siguientes líneas:

```
Platform.userExtensionDir

Platform.systemExtensionDir

File.mkdir(Platform.userExtensionDir)
```

Luego de configurar SuperCollider, se instala FoxDot (ver Tabla 1), una librería de Python para la creación músical, necesaria para la sincronía de Troop con SuperCollider.

```
$ pip install FoxDot
$ git clone https://github.com/Qirky/FoxDot.git
$ cd FoxDot
$ python setup.py install
```

Se abre nuevamente Supercollider y se ejecuta:

```
Quarks.install("FoxDot")
```

Cuando se han terminado de configurar estas herramientas se debe clonar Troop en el equipo:

```
$ git clone https://github.com/Qirky/Troop.git
$ git pull -a
```

Para usar Troop se teclea:

```
$ cd C:\Users\Guest\Troop
$ python run-client.py --mode SuperCollider
```

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

Posteriormente, se debe abrir Supercollider y ejecutar el siguiente código, como se muestra en la Figura 1:

```
Quarks.install("http://github.com/Qirky/TroopQuark.git")

Troop.start
```

Figura 1. Interfaz gráfica de SuperCollider

3.2. Características principales.

El proceso de instalación, descrito anteriormente, es el primer paso para realizar un análisis comparativo entre Estuary, Gibber y Troop. Teniendo en cuenta las diferentes características presentadas en la Tabla 2, tales como: sistema operativo, lenguajes soportados (DSL), interfaz gráfica, navegabilidad web y tipo de licencia, es posible hacer un acercamiento inicial a las plataformas. La plataforma SoOrOk se incluye en la Tabla 2, para visualizar sus características de diseño propuestas.

4. RESULTADOS

En esta sección se presenta un caso de uso en el que se analizan las diferentes sintaxis para los DSL soportados en cada plataforma. Además, se estima el tiempo de conexión entre la plataforma Site24X7 y los servidores web de Estuary y Gibber. Finalmente, se observan las estadísticas de sus contribuciones en GitHub.

4.1 Caso de uso.

El siguiente es un ejemplo de semántica de cada una de las plataformas analizadas, donde se podrán observar la sintaxis y estructura empleadas para generar el primer sonido.

Estuary.

Para usar esta herramienta de forma web se ingresa a https://estuary.mcmaster.ca/, o a https://127.0.0.1:8000 si se ejecuta localmente.

Al hacer clic en *collaborate*, se crea un nuevo ensamble y se escribe su nombre. A continuación, se escogen contraseñas diferentes para el administrador del ensamble, para los participantes que se van a conectar y para los usuarios que quieran ver sin hacer parte del performance. Por último, se selecciona el tiempo de expiración, es decir, cuánto tiempo va a durar el ensamble en la página oficial.

Luego de crear la sala o ensamble, como lo nombra Estuary, se accede a un tablero con seis espacios de trabajo. Cada espacio de trabajo puede seleccionar uno de los lenguajes disponibles en Estuary: MiniTidal (TidalCycles), Punctual o CineCer0 (CineVivo).

Para generar un sonido con MiniTidal se usa la siguiente sintaxis:

```
s "bd hh bd hh"
```

Donde la s simboliza que la línea de código será de sonido, y dentro de las comillas se escriben los sonidos deseados. MiniTidal tiene una lista enorme de sonidos ya predeterminada llamada Dirt-Samples (https://github.com/tidalcycles/Dirt-Samples).

Es importante mencionar que entre cada sonido debe haber un espacio ya que si se escriben dos sonidos juntos el sistema lo detecta como una sola palabra, buscando esta nueva palabra en la lista de sonidos; si coincide con algún nombre reproducirá el sonido y si no, no emitirá ninguno.

Si la sintaxis de MiniTidal parece un poco compleja, se puede usar CQenze, el cual es una alternativa de escritura de TidalCycles que separa los sonidos en

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

renglones para una visualización más agradable.

Con CQenze el código anterior se vería así:

```
bd :+-+-;
hh :-+-+;
```

Se escriben los sonidos en líneas separadas, luego dos puntos y a continuación + (más) si se quiere producir un sonido y - (menos) para que no suene, similar a la notación musical de negras y silencios.

Por otro lado, Estuary también permite la creación visual por medio de lenguajes como CineCer0 y Punctual. Se pueden crear figuras, formas y animaciones, como por ejemplo la creación de un círculo:

con Punctual

```
circle [-0.5,0.8] 0.75 >> rgb
```

y con CineCer0, en el que es necesario incluir la dirección web de la imagen que se va a incorporar

```
circleMask 0.5 $ vol 0.5 $ video "https://github.com/jac307/videoTextures/blob/master/otros/river.mov?raw=true"
```

Para ejecutar cualquiera de los códigos anteriores se debe dar clic en el triángulo verde que aparece al inicio de cada uno de los seis espacios de trabajo (ver Figura 2), en los cuales se puede seleccionar un lenguaje diferente y así complementar la creación artística de forma más integral, incorporando visuales y sonidos al mismo tiempo.

Figura 2. Interfaz gráfica de Estuary mostrando los seis espacios de trabajo

Tabla 2. Características de las plataformas de live coding

Plataforma	Sistema Operativo uso Web	Sistema Operativo uso Local	Lenguajes soportados (DSL)	Interfaz gráfica	Navegabilidad web	Tipo de licencia
Estuary	Microsoft Windows, Mac OS, Unix, Linux, Android.	Linux.	TidalCycles Punctual CineCer0 Hydra CQenze	Si	Si	GPL-3.0 License
Gibber	Microsoft Windows, Mac OS, Unix, Linux.	Microsoft Windows, Mac OS, Unix, Linux.	JavaScript OpenGL (glsl-fragment glsl-vertex) Hydra	Si	Si	MIT License
Troop	N/A	Mac OS, Unix, Linux.	TidalCycles SuperCollider Sonic Pi Python (FoxDot)	No	No	N/A
SoOrOk (Prototipo)	Microsoft Windows, Mac OS, Unix, Linux, Android	Microsoft Windows, Mac OS, Unix, Linux.	SuperCollider FoxDot	No	Si	GPL-3.0 License

Gibber.

Se ingresa a https://gibber.cc/alpha/playground/ para uso web o a https://127.0.0.1:8080 para uso local.

En la pantalla de inicio se encuentra la ayuda con algunos comandos básicos para usar la plataforma. Para realizar trabajo colaborativo, se debe dar clic en el botón Gibber, donde se ingresan los nombres del usuario y de la sala a la que se quiere ingresar. Se debe tener en cuenta que para trabajar en el mismo documento, todos los usuarios deben escribir el nombre de la sala exactamente igual incluyendo mayúsculas y caracteres especiales.

Al entrar a la sala se encuentra un chat y una hoja vacía donde todos los que ingresan podrán ver y editar. Dentro del mismo documento se pueden usar varios lenguajes tanto para video como para audio.

Para recrear el mismo patrón sonoro expuesto en Estuary, en Gibber se escribe en su espacio de trabajo, como se muestra en la Figura 3:

```
drums = EDrums()
drums.play( 'kd' )
drums.play( 'cp' )

s = Steps({
  kd: 'x.x.',
  cp: '.x.x'
}, drums )
```

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

El objeto `EDrums` contiene muchos de los sonidos de batería sintéticos inspirados en la Máquina de ritmos Roland TR-808 (Werner, 2014). Puede secuenciar los diferentes instrumentos refiriéndose a ellos con las abreviaturas `kd` (bombo), `sd` (caja), `ch` (platillo cerrado), `cp` (aplauso), `cb` (cencerro) y `oh` (platillo abierto).

Para crear los ritmos se usa `Steps` y luego a cada sonido se le asigna x para que suene y punto (.) para no generar sonido.

Al igual que Estuary, Gibber tiene la capacidad de crear visuales, animaciones o figuras usando la sintaxis de Hydra, otro DSL mostrado en la Tabla 1, así:

```
use( 'hydra' ).then( init => init() )

osc().out()

osc( ()=> Math.random() * 50 ).out()
```

Para ejecutar cualquier código en Gibber los usuarios deben seleccionar las líneas y usar el comando de teclado Ctrl + Enter.

Figura 3. Interfaz gráfica de Gibber mostrando su espacio de trabajo

Troop.

Para el uso de Troop se escribe el siguiente comando en la terminal, estando en el directorio donde éste se encuentra instalado:

```
python run-client.py --mode SuperCollider
```

Luego se abre SuperCollider y se escribe `Troop.start` y se usa el comando Ctrl + Enter para ejecutar esta línea.

Los usuarios que deseen colaborar en línea deben especificar la dirección IP `host=<hostname_or_ip_address>` y el puerto de acceso del usuario que será el servidor `port=<port_number>`, mediante:

```
python run-client.py -H host -P port
```

Se abre la terminal de Python (ver Figura 4), luego se escribe el siguiente código para realizar el mismo patrón musical de las plataformas anteriores:

```
p1 >> play("xoxo")
```

El método play contiene una lista de sonidos que se pueden acceder en: https://github.com/Qirky/FoxDot/tree/master/FoxDot/snd, donde cada carpeta está representada por una letra, que a su vez simboliza un sonido.

En un trabajo anterior (Molina et al., 2019), se evidenció que la visualización de los tiempos por medio de corchetes '[ ]' en los patrones, favorece la representación del código en forma de estructura musical. Es así como el código anterior queda:

```
p1 >> play("[x][o][x][o]")
```

Otra manera de copiar el mismo código pero separando los sonidos para tener un mejor control de cada uno de ellos es:

```
p1 >> play("[x][.][x][.]")
p2 >> play("[.][o][.][o]")
```

Cada línea se ejecuta usando Ctrl + Enter.

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

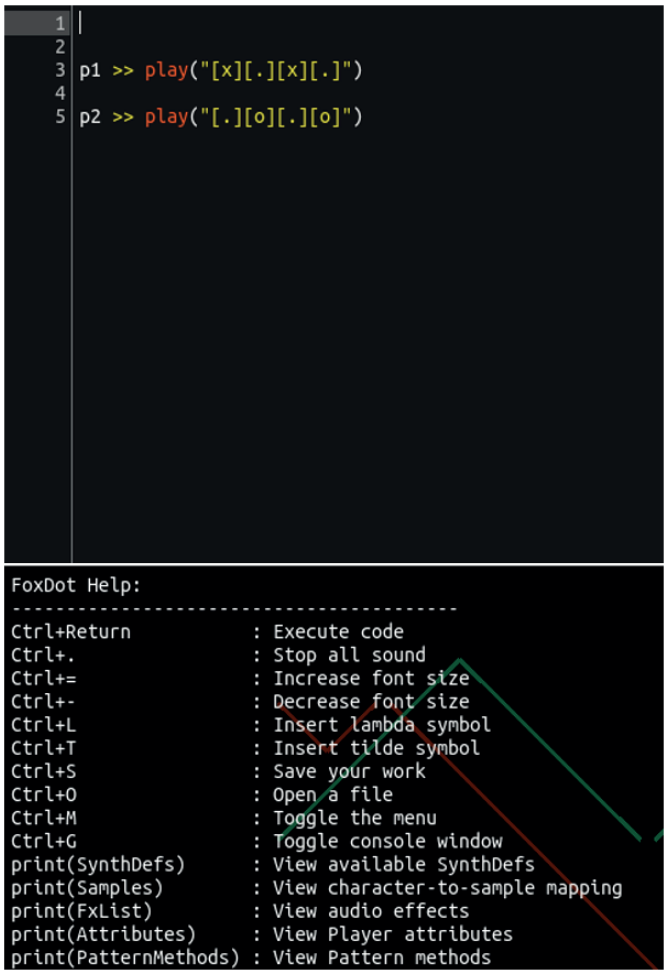


Figura 4. Interfaz gráfica de Python para trabajar FoxDot

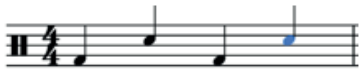
En la Tabla 3 se pueden comparar los códigos expuestos anteriormente. Adicionalmente, se muestra la representación en notación tradicional de los patrones. Los sonidos seleccionados son los más parecidos entre ellos: un bombo o kick y un redoblante o caja.

Tabla 3. Tabla de comparación de un patrón musical en las diferentes plataformas.

DSL y plataforma	Código
MiniTidal-Estuary	`s "bd cp bd cp"`
CQenze-Estuary	`bd :+-+-;` `cp :-+-+;`

js-Gibber	`drums = EDrums()`
	`drums.play( 'kd' )`
	`drums.play( 'cp' )`
	
	`s = Steps({`
	`kd: 'x.x.',`
	`cp: '.x.x'`
	`}, drums )`

FoxDot-Troop	`p1 >>`
	`play("[x][.][x][.]")`
	`p2 >>`
	`play("[.][o][.][o]")`

Notación musical tradicional de batería (Drums)	

4.2. Tiempo de conexión.

El tiempo de conexión corresponde al tiempo necesario para configurar la conexión entre la plataforma Site24X7 https://www.site24x7.com/ y los sitios web de Estuary y Gibber. Se usa Site24x7 porque es sencilla de usar y su versión gratuita no tiene límites de medición, requiriendo únicamente la dirección de la plataforma a analizar. Este mecanismo soporta peticiones HTTP y HTTPS.

En Troop no se mide el tiempo de conexión ya que no posee un servidor ni una página web, sino que los computadores interactúan directamente con sus IPs, convirtiéndolos en un servidor. Site24X7 no puede medir este tipo de conexiones.

Site24x7 siempre realiza el análisis en cuatro ubicaciones estándar: Toronto - CA, Fremont-CA - US, Melbourne - AUS y Singapore - SG. La Figura 5 muestra el tiempo de conexión entre estas cuatro ubicaciones y los servidores de Estuary y Gibber.

Los tiempos de carga inicial promedio de Estuary y

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

Gibber son: 0.6 s y 3.7 s, respectivamente. Las direcciones IP son: estuary.rhpcs.mcmaster.-ca./130.113.48.16 y 159.89.91.101, para Estuary y Gibber, respectivamente.

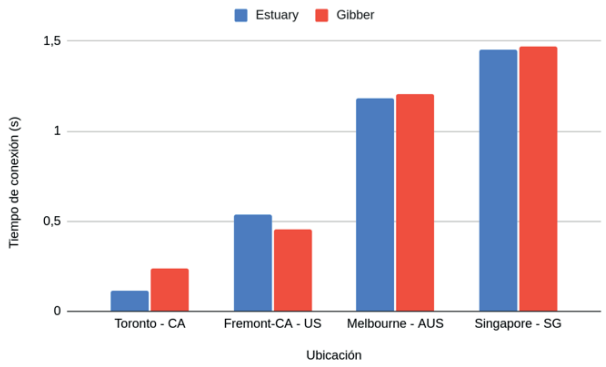


Figura 5. Tiempo de conexión entre cuatro ciudades al servidor de las plataformas Estuary y Gibber.

4.3. Estadísticas en GitHub.

La Tabla 4 presenta la calificación en votos positivos, los lenguajes base con los que fue construida y su porcentaje, el año de inicio y el enlace a la gráfica de las contribuciones desde su inicio, de acuerdo con la información del perfil de GitHub de cada una de las plataformas.

Tabla 4. Estadísticas en GitHub para cada una de las plataformas.

Plataforma	Votos en GitHub	Lenguajes base	Año de inicio	Gráfica de contribuciones
Estuary	103	Haskell 91,6% CSS 4,8% Nix 1,3% JavaScript 1,2% Otros 1,1%	2016	https://github.com/dktr0/estuary/graphs/contributors
Gibber	760	JavaScript 98,9% Otros 1,1%	2012	https://github.com/gibber-cc/gibber/graphs/contributors
Troop	249	Python 100,0%	2017	https://github.com/Qirky/Troop/graphs/contributors

4.4. SoOrOk.

Se propone SoOrOk (Sonatina para Ordenadores y Orquestas), una herramienta de colaboración en tiempo real que permitirá el live coding grupal en el mismo documento en múltiples computadoras. SoOrOk no es un lenguaje para la codificación en vivo, sino una plataforma para conectar varios live coders (músicos que programan en vivo) a través de una red.

Es una herramienta *plug and play*, es decir, el músico sólo tendrá que ingresar a la plataforma con una identificación sin necesidad de instalar ninguna aplicación.

SoOrOk se desarrollará de tal manera que sea posible acceder a través de un navegador compatible con el estándar HTML5. Su diseño es minimalista, sin poseer interfaz gráfica y su acceso se realizará por medio de una terminal.

Las funcionalidades principales de SoOrOk serán:

- Módulo intérprete musical:
  - Login o ingreso a sala.
  - Escritura de patrones en FoxDot.
  - Escucha de audio.
  - Visualización de la terminal de SuperCollider.
  - Visualización de niveles de audio de cada usuario.

- Módulo administrador
  - Recepción de correo electrónico con el cambio del nombre de la sala y contraseña.

Actualmente, SoOrOk se encuentra en fase de desarrollo. Su perfil de GitHub se encuentra en: https://github.com/diegomolinaquintero/SoOrOk.git.

5. DISCUSIÓN

Los DSL usados para el *live coding* se pueden instalar en los sistemas operativos más utilizados actualmente, como se observa en la Tabla 1.

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

A la fecha, no se cuenta con un archivo ejecutable que permita instalar las plataformas analizadas de forma guiada.

Siguiendo lo presentado en la sección 3.1, la instalación de Estuary es la más compleja ya que está construida con más lenguajes base. Gibber y Troop tienen procesos de instalación más sencillos aunque no por eso su uso es más simple.

Analizando los resultados presentados en la Tabla 2, se puede observar que, desde el punto de vista de uso web, Estuary posee la ventaja sobre Gibber de funcionar en dispositivos móviles. Sin embargo, ante la posibilidad de instalación de las plataformas en un sistema local, se evidencia que Gibber funciona en todos los sistemas operativos mientras que las otras dos están disponibles únicamente en sistemas basados en Unix.

Las interfaces web de Estuary y Gibber contienen múltiples secciones o capas donde se pueden encontrar desde tutoriales hasta información de los creadores. Al disponer de servidores propios de acceso gratuito, Estuary y Gibber no necesitan ser instaladas para ser usadas. No obstante, dado que dependen de un servidor central, no funcionarán si éste está caído o en mantenimiento. Troop, por su parte, necesita estar instalado en cada uno de los computadores donde se va a usar.

Estuary con cinco DSL, es la plataforma que soporta la mayor cantidad de lenguajes para el live coding simultáneamente, seguida de Gibber con una cantidad de cuatro. Aunque Troop permite el uso de cuatro DSL diferentes, sólo puede usar uno al mismo tiempo.

En cuanto al tipo de licencia, se observa que Estuary, al poseer una licencia GPL-3.0, permite usar, estudiar, copiar y modificar el código fuente de manera libre, mientras que la licencia de Gibber impone algunas limitaciones a la hora de reutilizar y compartir el código. Por lo tanto, se debe tener precaución a la hora de compartir alguna modificación en Gibber.

El caso de uso presentado en la sección 4.1 demuestra que al utilizar las plataformas analizadas de forma local, se requiere un nivel alto de programación, el uso de gestores de paquetes y el manejo de la consola (Terminal). Gibber y Troop permiten el uso de atajos de teclado para ejecutar líneas de código usando Ctrl + Enter, y Ctrl + . (punto) para detener el sonido, mientras que en Estuary se debe usar la interfaz gráfica para ejecutar los códigos. Aunque la sintaxis de los DSL es diferente, se puede observar en la Tabla 3 que el uso de múltiples símbolos tales como comillas, corchetes, más, menos y punto favorece la visualización de los patrones musicales.

En un trabajo previo (Molina et al., 2019) se desarrolló un método para convertir notación musical a una estructura de datos tipo tupla que puede ser leído por diferentes lenguajes de programación como Python, JavaScript o los DSL presentados en la Tabla 1. Esta característica forma parte del diseño de SoOrOk y la convierte en una opción muy potente para el desarrollo de *live coding.*

Troop permite la síntesis de audio, es decir, la manipulación de ondas para la creación de nuevos sonidos: Cambiar los archivos de audio por unos personalizados, crear sintetizadores propios en SuperCollider y manipular la salida de audio con plugins y mecanismos como Alsamixer, Jack, Jack2 o Cadence, y guardar los códigos. Por el contrario, Estuary o Gibber utilizan librerías de sonido tales como: Dirt-Samples y la digitalización de la TR-808, respectivamente.

Para el trabajo colaborativo, Troop es la más complicada de todas las herramientas analizadas, ya que requiere la ejecución de dos herramientas al mismo tiempo, en un orden predefinido, y encontrar y compartir la dirección IP. Comparado con abrir un navegador web, lo anterior puede convertirse en una tarea que implica mucho tiempo.

De acuerdo con la Figura 5, el tiempo de conexión es mayor para ubicaciones lejos del servidor tales como Melbourne y Singapur. No se observa una diferencia significativa entre las dos plataformas analizadas. El tiempo de carga inicial de Gibber es mucho mayor que el de Estuary ya que su plataforma web contiene más elementos de estilo CSS y JS, convirtiéndola en una plataforma *responsive* o

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

adaptativa, es decir, que permite una visualización correcta en distintos dispositivos.

De acuerdo a la información suministrada por el perfil de GitHub de cada plataforma (ver Tabla 4), las contribuciones de Estuary desde su creación se han mantenido estables, la comunidad participa activamente y se ve reflejado en la cantidad de aportes por parte de varios usuarios. Gibber por su parte ha tenido años completos sin contribuciones, actualizaciones o modificaciones. Aunque Troop es la más reciente entre las plataformas analizadas, su uso por parte de la comunidad se ha mantenido constante; sin embargo, en el último año sus contribuciones han disminuido notablemente.

6. CONCLUSIONES

Estuary está limitada a seis espacios de trabajo e incrementarlo sería un trabajo técnico arduo, además posee una sonoridad limitada que impide al músico personalizar de forma detallada sus sonidos, usando siempre los sonidos propuestos por la plataforma. Se recomienda el uso de esta plataforma para usuarios que inician en el *live coding.*

Gibber está en el punto intermedio entre las plataformas analizadas, permitiendo un uso ágil en la web y la creación de sonidos a partir de un banco de archivos sonoros mucho más extenso que Estuary, pero la falta de personalización de éstos crea un límite a la hora de integrarlo a una orquesta de computadoras. Se aconseja usar Gibber para presentaciones rápidas, ensambles esporádicos y usuarios con conocimientos básicos de *live coding.*

Troop permite la personalización y creación de sonidos muy detallados pero su falta de interfaz web y su compleja forma de conectar con otros usuarios, la convierte en una opción no recomendada para personas sin experiencia en informática. Por lo tanto, el uso de Troop se sugiere a usuarios con conocimientos avanzados en el *live coding.*

El uso de los navegadores web para la práctica colaborativa del *live coding* ahorra tiempo a los intérpretes, ya que evita la realización de instalaciones complejas. De esta manera, un músico que no conozca de sistemas puede empezar a crear o componer al instante, usando un navegador web tradicional. Esto permite preocuparse más por la creación artística que por el conocimiento técnico en informática. Para músicos que quieran iniciar en el mundo de las artes digitales escribiendo código, el uso de DSL y herramientas como las expuestas puede ser una excelente iniciación.

El trabajo local de todas las plataformas requiere un nivel medio-alto en conocimientos técnicos. Se trabaja localmente cuando se requiere modificar una parte específica de la interfaz de usuario, agregar funcionalidades nuevas o simplemente conectar presencialmente varios dispositivos en una misma red.

La facilidad de conectar usuarios desde cualquier lugar del mundo para interpretar live coding de manera colaborativa, proporciona experiencias de intercambio cultural, crea nuevos conceptos creativos a partir de culturas, experiencias y estilos de trabajo diferentes.

Se propone la creación de una plataforma de *live coding* colaborativo que permita su utilización en un entorno web, facilitando el acceso de usuarios no expertos en programación. Esta herramienta se denomina SoOrOk y está diseñada tanto para músicos que deseen crear sus propios sonidos como para informáticos que quieran experimentar con la música sin necesidad de tener nociones musicales.

ISSN: 2711-3760
DOI: https://doi.org/10.47185/27113760.v2n2.68

REFERENCIAS

Aaron, S., & Blackwell, A. F. (2013, September). From sonic Pi to overtone: creative musical experiences with domain-specific and functional languages. In Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling & design (pp. 35-46).

Betancur, E. (2016). Diseño e implementación de un DSL: CQenze, como lenguaje de primera experiencia para el código en vivo. In Proceedings of the Festival Internacional de la Imagen 2015, 1-6.

Collins, N., McLean, A., Rohrhuber, J., & Ward, A. (2003). Live coding in laptop performance. Organised sound, 8(3), 321-330.

Devopedia. 2020. "Package Manager." Version 7, July 22. Accessed 2021-09-09. https://devopedia.org/package-manager.

Kirkbride, R. (2016, October). Foxdot: Live coding with python and supercollider. In Proceedings of the International Conference on Live Interfaces (pp. 194-198).

Kirkbride, R. (2017). Troop: a collaborative tool for live coding. In Proceedings of the 14th Sound and Music Computing Conference (pp. 104-9).

Littler, C., Ogborn, D., & Sicchio, K. (2021). JSoLangs: ephemeral esolangs in a collaborative live coding environment. Obtenido de: http://dx.doi.org/10.17613/yq3a-1s78

McKinney, C. (2014, June). Quick Live Coding Collaboration In The Web Browser. In NIME (pp. 379-382).

McLean, A. (2014, September). Making programming languages to dance to: live coding with tidal. In Proceedings of the 2nd ACM SIGPLAN international workshop on Functional art, music, modeling & design (pp. 63-70).

Molina-Quintero, D., Betancur-Gutiérrez, E., & Benítez-Tamayo, S. (2019). Diseño e implementación de notaciones alternativas para la informática musical. Proceedings of the Festival Internacional de la Imagen 2019, (pp. 316-326).

Trejos, O., & Muñoz, L. (2020). Introducción a la programación con Python. Bogotá, Colombia: Ediciones de la U.

Trueman, D. (2007). Why a laptop orchestra?. Organised Sound, 12(2), 171-179.

Magnusson, T. Herding Cats: Observing Live Coding in the Wild. Paper, Falmer, Brighton, BNI 9QJ, UK.

Ogborn, D., Beverley, J., del Angel, L. N., Tsabary, E., & McLean, A. (2017). Estuary: Browser-based collaborative projectional live coding of musical patterns. In Proceedings of the 2017 International Conference on Live Coding.

Roberts, C., & Kuchera-Morin, J. (2012). Gibber: Live coding audio in the browser. In ICMC (Vol. 11, p. 6).

Rodríguez, J., Betancur, E., & Rodríguez, R. (2019). CineVivo: a mini-language for live-visuals. In Proceedings of the 2019 International Conference on Live Coding.

Sedláková, K. (2021). Vizuálny live coding ako nástroj formujúci estetický prežitok: Hydra vs. p5. js (Doctoral dissertation, Masaryk University, Faculty of Arts).

Sorensen, A., Swift, B., & Riddell, A. (2014). The many meanings of live coding. Computer Music Journal, 38(1), 65-76.

Werner, K. (2014). The TR-808 Drum Machine and its Emulations. University of California, Berkeley, 24, 26.

Wilson, S., Cottle, D., & Collins, N. (2011). The SuperCollider Book. The MIT Press.