Universidad Nacional de Rosario
Facultad de Humanidades y Artes
Escuela de Música

Ampliación y optimización de consolas de audio digital para sonido en vivo: desarrollo de una aplicación en *Pure Data*

Seminario de Investigación
Informe final

Autora: Basso, Catalina.

Profesor/Tutor: Buján, Federico.

Carrera: Licenciatura en tecnologías aplicadas al arte sonoro.

2025

Índice

Acerca del estudio

Acerca del estudio

Justificación del proyecto y situación problemática: respondiendo a demandas de eficiencia y flexibilidad en un ámbito cambiante

La industria del sonido en vivo es un campo profesional donde la única constante es el cambio. Como en él conviven dos áreas —arte y ciencia— que ya de manera independiente están en constante movimiento no resulta extraño que la conjunción de las mismas dé como resultado algo muy vertiginoso. (Scheirman, 2013). A su vez, los avances tecnológicos acontecen cada vez de manera más acelerada: “la aceleración del cambio en nuestro tiempo es, en sí misma, una fuerza elemental” (Toffer, 1970) y muchos de los dispositivos que aún hoy se utilizan suelen quedar obsoletos. Las formas que encuentran los profesionales de hacer frente a esta situación pueden ser muy variadas: las grandes empresas pueden optar por cambiar los equipos con más regularidad, ya que cuentan con un gran capital de inversión pero esta rápida tasa de obsolescencia tecnológica conlleva una gran desventaja para los competidores más pequeños, quienes deben buscar estrategias adaptativas para seguir siendo relevantes en un mercado difícil. La creatividad juega un papel fundamental para afrontar dichas limitaciones técnicas y existen diversas maneras de solventar la problemática y que la experiencia performática musical no sufra sus estragos.

El presente trabajo se propuso ofrecer una respuesta a la problemática de la obsolescencia funcional en dispositivos de audio todavía ampliamente presentes en el mercado y así realizar un aporte a la búsqueda de soluciones. El enfoque estuvo aplicado al ámbito del sonido en vivo, donde el principal *hardware* de manipulación de audio son las consolas o mesas de mezcla, las cuales reciben las señales eléctricas provenientes de las fuentes sonoras y a su salida las señales alteradas son enviadas a los emisores o parlantes. Los procesamientos que pueden aplicarse a las señales pueden ser tanto dentro del dominio analógico como del digital: por ende podemos decir que las consolas de audio se dividen en dos grandes grupos, aquellas donde sólo hay señales analógicas y aquellas que poseen conversores analógico-digitales en sus entradas y digitales-analógicos en sus salidas para así procesar la señal como información binaria, es decir de manera digital. Aunque las

mesas de mezcla analógicas no han desaparecido e incluso muchos artistas y sus operadores las eligen por su cualidad sonora nuestro desarrollo apuntó a las consolas digitales, debido a su mayor potencial de integración y expansión. A diferencia de las consolas analógicas, las consolas digitales permiten establecer comunicación con dispositivos externos —computadoras, controladores o sistemas de automatización— mediante protocolos de comunicación como MIDI. Esta capacidad de interpretación de información de control habilita la incorporación de procesos adicionales que no forman parte del diseño original del hardware.

En este contexto, el objetivo del presente estudio fue desarrollar una herramienta que amplíe las prestaciones de las consolas digitales, sorteando sus limitaciones nativas y ofreciendo al operador de sonido una mayor flexibilidad operativa y creativa en entornos de performance en vivo, ofreciendo una herramienta de trabajo basada en *software*, versátil y estratégica: un recurso competente cuyo rendimiento esté alineado con los requerimientos de las presentaciones en vivo, estable y eficiente. La propuesta apuntó a realizar un producto que logre ser un complemento para algunas de las consolas digitales que, aunque han salido modelos nuevos y más caros con las funcionalidades que anteriormente estaban carentes ahora presentes, permanecen relevantes ya que están hoy todavía muy presentes en el mercado, principalmente en locaciones de menor escala y presupuesto.

Se desarrolló en el entorno gráfico de programación *Pure Data (PD)* una aplicación orientada al usuario que comunique a la consola con un dispositivo externo que aloje al *software.* La elección del lenguaje radicó en la necesidad de que la implementación de este producto sea, en principio, accesible para el usuario y *Pure Data* es de licencia libre, lo cual significa que cualquiera puede descargarlo y usarlo sin restricciones de ningún tipo. Además es un programa con baja demanda de CPU y que no ocupa en exceso la memoria de la computadora donde se aloje e incluso los archivos son muy ligeros, de unos pocos megas en general. En la industria y en la academia PD es usado para prototipar ya que, al ser un entorno de alto nivel, rápidamente se pueden lograr resultados pero además porque no permite compilar el *patch* o código fuente en una aplicación ejecutable. Para eso se necesitaría migrar el trabajo a *Plug Data* o directamente programar el código en un lenguaje como C++. Para el caso de este trabajo no se vió necesario este procedimiento ya que no se buscó utilizar el programa como *plugin* dentro de un

DAW ni como aplicación *standalone* sino que se invita al operador a usar directamente el *patch* de *Pure Data*, permitiendo así una mayor flexibilidad para usuarios experimentados y la posibilidad de futuras expansiones colaborativas. Por último, PD fue originalmente diseñado para procesar señales de audio en tiempo real siendo así utilizado por artistas experimentales y como luthería digital para *shows* en vivo, lo cual nos brinda la capacidad de realizar procesamientos instantáneos sin demoras.

En cuanto al criterio de selección de las problemáticas específicas dentro de la temática que se buscan abordar —las operaciones que se intentarán tercerizar— en principio lo más importante fue la pertinencia. Desde la experiencia personal como profesional dentro de la industria del sonido en vivo, tanto como desde el rol de *stage manager* (encargado de escenario) asistiendo a los operadores, como vivencias directas operando, se ha ido observando con el tiempo cuáles son las principales carencias de algunos modelos de consolas digitales ampliamente utilizados en la ciudad de Rosario, particularmente la consola digital de Yamaha, modelo LS-9. De este modo, la relevancia de los módulos que se buscan programar en esta propuesta de investigación de desarrollo para el rubro profesional se vió justificada directamente por la *praxis*, hallamos a través de la propia práctica musical la situación problemática aquí expuesta y a partir de ésto se propuso el desarrollo de una herramienta que permita mayor libertad creativa en el uso de quienes manejan las consolas de audio: consideramos al operador de sonido un actor determinante en el resultado sonoro de una música performativa y valoramos enormemente su aporte estético y creativo, y por lo tanto este trabajo de desarrollo aplicado se propuso finalizar con una herramienta abierta y disponible para ser llevada a la práctica y brindar un aporte a la comunidad artística, enmarcando al trabajo dentro de la investigación artística en música. (López Cano, 2014).

Además, se tuvieron en cuenta dos requerimientos técnicos principales a la hora de pensar y diagramar la propuesta: la viabilidad y la replicabilidad. En cuanto a la viabilidad es sumamente necesario, al tratarse de una aplicación dentro de presentaciones en vivo, que haya una latencia suficientemente baja como para no ser percibida como un retraso temporal en la señal. Además se necesita que el programa sea de fácil implementación para el operador, que no conlleve la necesidad de una vasta experiencia en programación orientada al audio sino conocimientos básicos de informática y conexionado de dispositivos. Por último, en

cuanto a la replicabilidad se buscó que la transmisión de información entre la consola y el ordenador que aloje al *software* permita el aprovechamiento de esta herramienta para más de un modelo de consola y que invite a distintos usos de los agentes implicados, siendo flexible y personalizable.

Preguntas de investigación

¿Se puede desarrollar *software* que solvente las limitaciones que presentan las consolas de audio digital y extender nuevas formas de uso?

¿Cómo establecer una comunicación entre la consola y el ordenador eficiente para los usos requeridos? ¿Qué dispositivos, conexiones de *hardware* y configuraciones de *software* son requeridas?

¿Qué parámetros de la consola se pueden controlar con PD a través del lenguaje MIDI que lleven a una extensión de las capacidades de la consola? ¿Qué procesamientos puedo adicionar desde la aplicación? ¿Qué arquitectura conlleva esta aplicación dentro de PD?

¿Realizar este trabajo puede ayudar a construir una base metodológica tanto de investigación como de programación que permita futuros desarrollos aplicados para el sonido en vivo?

Antecedentes

En la literatura académica actual existen diversos artículos publicados que formalizan propuestas de desarrollo aplicado relevantes a la situación problemática planteada en este trabajo de investigación. En su mayoría están constituidos por controladores que buscan facilitar la interfaz de las consolas con un diseño HCI (*Human-Computer Interaction*), debido a que no se han realizado estudios de usabilidad relacionados a esos dispositivos y las investigaciones aquí citadas se preguntan si están optimizadas para la tarea que fueron hechas. En la *Conferencia Internacional de nuevas interfaces para la expresión musical* de 2011, en Oslo, Noruega, investigadores del *Music Technology Group* de la *Universitat Pompeu Fabra* presentaron un artículo en donde dan cuenta del desarrollo de una interfaz de control *multitouch*. Proponen un conjunto de controles que pueden ser usados para

simplificar o mejorar la *performance* en las tareas de mezcla de audio, tomando en consideración las tecnologías de mezcla modernas como el audio *surround* o 3D.

A nivel local, en la Universidad Nacional de Mar del Plata se realizó a modo de proyecto final de la carrera de ingeniería electrónica el “diseño, desarrollo y la construcción de un prototipo de un Controlador Remoto para ser utilizado con el protocolo digital de comunicaciones MIDI”. En este caso el trabajo está apuntado a mejorar la experiencia de mezclar en una computadora dentro de un *DAW,* donde “se emula a las antiguas consolas analógicas de grabación”, por lo que “la utilización de un controlador remoto se origina en la necesidad de facilitar y agilizar la operación de estas consolas virtuales de grabación, sustituyendo total o parcialmente el uso del mouse y del teclado con, por ejemplo, potenciómetros rotativos o deslizables, encoders, interruptores, conmutadores, botones, etc.” (Melczarsky, 2004). En los dos casos citados anteriormente las herramientas creadas no realizan procesamiento o cálculos ya que su única función es la de transmitir información de control para dirigir a la mesa de mezcla desde un hardware externo pero sin realizar un cómputo. (Carrascal, 2011).

Otro caso similar lo conforman las “*Stagemix”*: aplicaciones propias de los fabricantes de consolas que permiten al operador manejar parámetros de manera remota, lo cual es muy útil cuando el FOH (*front of house*) no se encuentra posicionado adecuadamente o cuando se quiere cambiar algo desde el escenario, lejos de la consola. Algunas versiones permiten incluso ejecutar múltiples aplicaciones en simultáneo en distintos dispositivos, de manera que cada músico maneje la mezcla de su monitor de retorno. Para este caso tampoco se requiere un procesamiento de audio o de datos, ya que funcionan exclusivamente como controladores. Algunos ejemplos de estas aplicaciones comerciales son la *Yamaha StageMix*, *Digital Mixers for iPad/Android Tablets* de Behringer y la *SQ Mixpad* de Allen & Heath.

Por último, algunos desarrolladores han creado aplicaciones en donde sí se añaden funcionalidades a las consolas de audio digital para uso en vivo por medio de procesamiento externo. Suele haber un enfoque particular en la incorporación de herramientas para paneo de sonido envolvente, posibilidad de posicionamiento de las fuentes en el espacio sonoro de manera multicanal (discreta o por codificación orientada a objetos) que las consolas no han incorporado todavía. Estas investigaciones suelen estar enmarcadas en el ámbito comercial y cuentan con

patrocinadores o financiamiento de algún tipo, por lo que la disponibilidad de información detallada sobre la arquitectura de dichos programas es limitada, como de igual manera sucede con las aplicaciones de control remoto de las marcas de consolas. Para este caso es de particular interés el trabajo de Ianina Canalis, especialista en audio espacial en Meyer Sound quien desarrolló el ISSP (Immersive Sound System Panning), un programa que permite realizar paneo inmersivo en vivo.

Hipótesis

El desarrollo e implementación de una herramienta realizada en *Pure Data* permite extender las capacidades de control de consolas de audio digital mediante el agrupamiento de canales en grupos de control virtuales (DCA), optimizando la gestión de niveles en situaciones de sonido en vivo. Un *patch* dinámico, modular y escalable —basado en la creación y modificación en tiempo real de objetos y conexiones dentro de subpatches y en la clonación de abstracciones— incrementa la adaptabilidad de la propuesta y permite reorganizar el flujo de datos de manera eficiente y a demanda de las necesidades operativas particulares.

La transmisión de información entre los dispositivos se construye a partir de mensajes MIDI del tipo no registrados (NRPN) —de implementación rápida aunque baja replicabilidad debido a las variaciones entre modelos de consola—. Dada la estructura de control de las consolas se debe trabajar con paquetes de cuatro mensajes para representar un único parámetro, lo cual justifica la necesidad de un *software* que opere de intermediario entre *Pure Data* y la salida MIDI del ordenador para codificar correctamente el retorno de la posición actualizada de los *faders*.

Una segunda estrategia de transmisión de información contempla el envío y recepción de señales entre ambos dispositivos para control y procesamiento de audio en tiempo real. Aunque esta opción abre el abanico de prestaciones y habilita posibilidades creativas interesantes, los niveles de latencia asociados a la transmisión de audio en un sistema de esta índole limitan su eficacia para tareas de carácter inmediato, restringiendo su aplicación en un entorno operativo. En conjunto, estas afirmaciones permiten concluir que la arquitectura del software debe asentarse exclusivamente en el intercambio de mensajes MIDI del tipo NRPN:

estrategia que ofrece la respuesta temporal, la estabilidad y el grado de control necesarios para ampliar las prestaciones de la consola.

Objetivos

Objetivo general

En este trabajo de investigación de desarrollo aplicado nos proponemos desarrollar un *patch* (o programa) en *Pure Data* en donde la computadora funcione no sólo como controlador externo, con la posibilidad de recibir y enviar información MIDI, sino que a su vez realice los procesamientos necesarios para el uso requerido. El objetivo principal es el de brindar al usuario una herramienta útil para extender las prestaciones de las mesas de mezcla y solventar sus limitaciones de diseño.

Objetivos específicos

1. Generar DCA virtuales (*Digital Controlled Amplifier*): controlar *faders* (deslizadores) por grupos manteniendo el nivel relativo de cada canal.
2. Generar automatizaciones para diversos usos.
3. Lograr una latencia lo suficientemente baja que no sea percibida.
4. Desarrollar el programa de tal modo que la experiencia sea flexible y personalizable para cada operador de sonido que desee utilizarlo.
5. Establecer una base normativa de comunicación consola-pc (y PD) para poder generar a futuro nuevas aplicaciones o nuevas extensiones dentro del mismo programa.

Estrategia metodológica: ¿cómo abordar una investigación de desarrollo aplicado?

Tipo de estudio

El presente trabajo se constituyó como una investigación de desarrollo aplicado, el cual conllevó la creación de un producto que busca abordar una problemática particular. Para esto se programó una herramienta de software en el entorno de programación visual de *Pure Data* y se buscó generar estrategias de uso

y de comunicación entre los dispositivos de hardware para ofrecer al operador de sonido en vivo un instrumento de trabajo innovador y principalmente que permita la implementación de nuevas ideas. La propuesta incluyó realizar un marco teórico sólido para consolidar el conocimiento alrededor de la temática, agrupando diferentes dimensiones de la misma, una recopilación de antecedentes para reunir trabajos similares como referencia metodológica y por último una documentación exhaustiva del proceso de programación, en donde se de cuenta de las diferentes fases del trabajo investigativo, buscando trazabilidad.

Fundamentación metodológica de la praxis

El matemático húngaro George Polya empieza su libro *Cómo plantear y resolver problemas*, donde proporciona heurísticas generales para resolver problemas de todo tipo, con la siguiente frase: “Un gran descubrimiento resuelve un gran problema, pero en la solución de todo problema, hay un cierto descubrimiento.” (Polya, 1965). De acuerdo con Polya y entendiendo a la heurística como aquella “ciencia con el objeto de estudiar las reglas y los métodos del descubrimiento y de la invención”, resolver un problema requiere recuperar conocimientos previos, establecer analogías, generalizar o particularizar elementos y, en muchos casos, descomponer el problema original en subproblemas más accesibles. (Polya, 1989). Este enfoque resultó particularmente pertinente para el desarrollo del sistema en Pure Data, donde la funcionalidad se alcanzó mediante ciclos iterativos de prueba, variación y refinamiento. El proceso no se estructura de forma lineal sino a través de la formulación, variación y resolución progresiva de problemas técnicos por etapas, probando, evaluando y corrigiendo en cada vuelta, donde cada avance implica cierto grado de descubrimiento.

Siguiendo a Polya, la resolución de un problema depende de la capacidad de relacionarlo con otros ya resueltos, y este principio resultó crucial para este trabajo. El diseño de los módulos, el uso del protocolo de comunicación MIDI, el control de DCAs virtuales (externos), y las estructuras internas del patch en Pure Data se apoyaron en casos previos: antecedentes de propuestas de desarrollo aplicado similares que fueron relevadas y estudiada, dispositivos MIDI comerciales así como también desarrollos propios anteriores. A partir del relevamiento de la literatura actual se ha concluido que las propuestas en general abundan en cuanto al

desarrollo de controladores pero hay carencia de investigaciones en cuanto a dispositivos que realicen procesamiento de las señales o de datos y no sólo control externo. Se espera poder llenar esa vacancia a partir de esta investigación y que la misma sirva como punto de partida para próximas propuestas similares.

En cuanto a las fases del desarrollo se siguió el concepto de Polya de distinguir entre “concepción del plan” y “ejecución” para así aceptar en las primeras fases aproximaciones parcialmente funcionales, útiles como andamiajes conceptuales para guiar el proceso y en los operativos finales recién exigir rigor operativo que asegure la estabilidad y correcta aplicación del dispositivo. Además, se tomó del mismo autor la idea de “variación del problema” como parte central de la praxis: cada módulo se concibió inicialmente como un “problema auxiliar”, más acotado y accesible, cuya solución habilitaba la integración posterior en un sistema mayor. Esta descomposición respondió a una lógica de ingeniería iterativa más que a una verificación experimental, dado que el objetivo principal era alcanzar funcionalidad, estabilidad y operatividad en el contexto del sonido en vivo.

Por último, siendo que esta investigación no es solo de desarrollo sino que la propuesta está situada dentro de un campo determinado, en este caso el sonido en vivo, donde se busca resolver un problema concreto aplicado, es necesario determinar cómo se ven justificadas las supuestas problemáticas del campo. Las demandas de los operadores para las cuales en este trabajo se busca brindar una solución se ven justificadas por la experiencia personal dentro del rubro profesional específico, enmarcando al trabajo dentro de la “investigación artística en música” como la plantea el musicólogo Rubén Lopez Cano en su obra. (Lopez Cano, 2014). La validez del trabajo no radica en la formulación de modelos teóricos abstractos, sino en la capacidad del sistema desarrollado para operar de manera eficaz, estable y pertinente dentro de los contextos profesionales particulares.

Fases del proyecto

A continuación daremos una breve reseña de cómo el trabajo fue evolucionando. Como ya se ha mencionado, esta investigación tiene dos soportes principales: el sustento teórico que fundamenta al desarrollo y las decisiones técnicas, para el cual se realiza una revisión bibliográfica de materiales pertinentes,

y en segundo lugar el relevamiento de la literatura existente, a partir de la cual se establece un estado del arte y antecedentes concretos de otras investigaciones de desarrollo aplicado relacionadas y se hace análisis de los mismos para conocer cómo afrontan en cada caso la situación problemática.

Lo primero que se buscó fue identificar las problemáticas específicas dentro de la situación-problema para tener una primera aproximación a qué se quiere programar, principalmente haciendo uso de los antecedentes relevados y de la experiencia personal dentro del rubro. Además, se engrosó la base teórica de la investigación por medio de fuentes de primera mano como manuales de usuario así como con artículos y libros publicados que fuesen relevantes a la temática.

En segundo lugar se realizó un primer esquema especulativo del programa utilizando la bibliografía específica de Pure Data y del funcionamiento general del sistema planteado a partir de lo relevado para plantear posibles caminos a tomar. Luego se comenzó a desarrollar el primer prototipo. Se programaron los módulos correspondientes y se simuló el uso de la consola con la *DAW Reaper*: comunicación a través de un cable virtual MIDI gratuito (*LoopBeInternalMIDI*). Dentro del proceso se optó por cambiar el método de programación dentro del entorno de estático a dinámico. Esto permite una capacidad de toma de decisiones mucho más en detalle y personalizadas para el usuario. Se cuenta entonces con una cara visible donde se interactúa con el programa y se le indica cómo proceder y una trasera en donde se van creando objetos y conectando a los mismos de manera que cumplan con el objetivo propuesto.

Posteriormente, con el primer prototipo terminado se realizaron pruebas piloto de comunicación consola-pc para llevar un registro detallado de los resultados y analizar errores o áreas de posible mejora. Luego de varios prototipos se llegó a una versión funcional, la cual fue sometida a un control general y perfeccionamiento del código y se diseñó la interfaz destinada al usuario buscando una estética amigable y entendible. Una breve descripción del programa e instrucciones de uso son adjuntas a modo de comentarios dentro del *patch*. Por último se analizan los resultados para dar cuenta qué pudo ser cumplido dentro de los objetivos iniciales y si los supuestos iniciales fueron confirmados y se concluye sobre el aporte del trabajo y futuras posibilidades.

Estrategias de desarrollo: protocolos de comunicación y arquitecturas de programación

El sistema que se plantea cuenta con dos dispositivos interconectados la consola de audio digital y el ordenador que aloje a la aplicación, por lo que se debe buscar una manera en la que el vínculo sea efectivo en cuanto a la replicabilidad, la facilidad de implementación, la baja latencia y el potencial del proyecto. Una vez relevados los antecedentes de investigaciones similares se llega a la conclusión de que hay dos tipos de comunicación posibles entre la mesa de mezcla y la computadora: la transmisión de audio y la transmisión de información MIDI. La última puede también ser subdividida en dos: configuración de parámetros pre-seteados y el mapeo MIDI personalizado. Para comprender las distintas estrategias en profundidad se despliega en el primer capítulo de esta investigación un marco conceptual esclarecedor.

En cuanto a los puntos fuertes y débiles de los dos tipos de transmisión de información MIDI se puede decir que el uso de los números de parámetros pre existentes en las consolas cuenta con la ventaja de consumir una muy baja cantidad de recursos, ser un estándar ampliamente soportado para el control de dispositivos de audio y facilitar la transmisión de mensajes específicos. Una desventaja de este tipo de mensaje MIDI radica en que el control de parámetros y ajustes predefinidos puede ser más limitado y el principal punto en contra es que aunque algunos fabricantes intentan mantener consistencia para facilitar la integración no existe un estándar universal para los valores de cada parámetro.

Con la segunda opción dentro de la transmisión de información MIDI, el mapeo personalizado, se logra una alta flexibilidad y personalización en cuanto a los parámetros que controla el programa y así permite adaptarse a las necesidades específicas del usuario. Además, puede aplicarse a cualquier consola de audio digital ya que los identificadores utilizados dejan de ser los preestablecidos. Sin embargo, implica una implementación más compleja que demanda tiempo para el usuario.

Por último, en cuanto a la transmisión de señales de audio se entiende que sería una opción muy interesante que agregaría al trabajo muchas prestaciones pero su implementación es bastante más compleja. Permite el procesamiento de audio en tiempo real, la posibilidad de implementar efectos y procesamientos

digitales avanzados, y controlar las señales, monitorizarlas, realizar análisis de espectro y demás. Sin embargo, esto requiere una mayor capacidad de procesamiento, consume más recursos de hardware, la implementación es distinta ya que requiere de diferentes conexiones y se prevé que es muy probable que presente problemas importantes de latencia.

En cuanto a la arquitectura de programación utilizada en *Pure Data* podemos dar cuenta de dos grandes maneras de trabajar: de manera estática y de manera dinámica. La programación estática implica que el usuario recibe algo pre-programado en donde puede elegir entre ciertas opciones que se le brindan pero éstas son limitadas, no hay flexibilidad en cuanto a lo que el operador puede hacer con la herramienta sino que debe atenerse a lo que el desarrollador del programa realizó con anterioridad. En cambio, si se trabaja con constructores se brinda la capacidad de programar en tiempo real. Es decir, el usuario toma las decisiones pertinentes (por ejemplo, cuántos grupos de DCA necesita y cómo quiere agrupar los canales) y el programa responde de manera que esas decisiones son llevadas a cabo, creando los objetos necesarios y conectando *inlets* y *outlets* según el orden de creación de los elementos. Se buscará contar con esta última opción para que la aplicación sea personalizable y pueda adaptarse a distintas situaciones y formaciones musicales.

Consideraciones previas de la arquitectura de la aplicación

Desde un comienzo se pensó que para la implementación de DCA virtuales se debía recibir de la consola las posiciones originales de cada *fader* por separado para mantener las relaciones de nivel originales entre canales a la hora de incrementar o decrementar el nivel del grupo. Luego, las posiciones actualizadas deberían poder ser enviadas de vuelta a la consola y ser interpretadas por la misma para mover el *fader* al nuevo valor correspondiente de manera automática. Para permitir la comunicación entre la computadora que aloja el *patch* de *Pure Data* y la mesa de mezcla cuyas prestaciones se buscan optimizar se encontró que sería necesario conocer en qué valores de *Control Change* o *controllers* se encuentran los parámetros que se quieren controlar de manera remota. Se pensó en una primera instancia que de manera directa con los objetos [ctlin] *(control in)* y [ctlout] *(control out)* sería posible visualizar un valor de control o de canal distinto que discriminase los *faders*. Posteriormente, en la segunda prueba dentro del trabajo de campo este

supuesto inicial se desmintió. En cuanto a la conexión física desde un primer momento se consideró el uso de un cable USB A/b (formato A para la pc y B para la consola) o del cable MIDI de 5 (cinco) pines.

Se anticipó que con consolas cuyos deslizadores no sean motorizados el programa será útil pero se necesitará una visualización de cambio de nivel mediante un vúmetro dentro de la consola. Para consolas analógicas la aplicación se previó no sería realizable ya que no se podría manejar información MIDI sino que se necesitaría enviar y recibir señales de audio para agruparlas y cambiar sus niveles con multiplicadores de señal, objetos del tipo [*~] en *Pure Data.* Se descartó en esta primera fase la idea por la problemática de la latencia en el envío y recepción de señales a diferencia de los datos.

Desde un primer momento se supo que en el caso de que dentro del programa se pueda contar con una instancia de programación dinámica se estaría aportando enormemente a la personalización y flexibilidad del mismo, del modo que cada operador con unos pocos clicks puede elegir cómo quiere agrupar los canales dentro de los grupos. Posibilidad lograda mediante mensajes que con el uso del punto y coma (;) hablan de manera remota al entorno y permiten la creación de objetos y la colección de objetos según su índice de creación e índices de *inlets* y *outlets.* Además, se supuso que con instancias de clonación, posibles gracias al objeto de PD [clone], la escalabilidad del proyecto se vería potenciada. Contando con esta opción ya no es necesario que manualmente se cree cada grupo sino que se designa la cantidad máxima posible y *clone* se encarga de reproducir una abstracción al estilo plantilla, o clase dentro de la programación orientada a objetos, la cantidad de veces indicada, siendo a su vez cada instancia independiente de las demás. Esto es particularmente pertinente ya que permite que la herramienta pase a ser útil incluso en la práctica con consolas que ya de por sí cuentan con DCA porque permiten un número casi ilimitado de grupos.

Sobre el recorte

Replicabilidad y elección de las nuevas prestaciones

La replicabilidad de la herramienta que se busca desarrollar corresponde a la variedad de marcas y modelos de consolas de audio digital para sonido en vivo en

los que sea posible y pertinente aplicar este producto. Se va a trabajar en el orden de las siguientes categorías ordenadoras que solventan las necesidades de los usuarios planteadas en la situación problemática. Como primera prestación se considera la generación de DCA (*Digital Controlled Amplifier*) para permitir crear grupos personalizables de control de amplitud, lo cual permite el control de varios faders por medio de uno “madre”, muy usado por ejemplo para agrupar todos los cuerpos de la batería o todos las voces que realizan coros. Esto mejora considerablemente la gestión de mezclas complejas y facilita el control de muchos canales en simultáneo.

En segundo lugar se piensa en incorporar automatizaciones. Es decir, permitirle al operador de audio en vivo delegar algunas de las acciones que deben realizarse con rapidez o en simultáneo, para así incrementar la eficiencia operativa y reducir el margen de error humano en entornos en vivo y permitir al operador la dedicación a tareas más creativas y de escucha atenta.

Primer capítulo
Marco teórico: fundamentos conceptuales y contextuales

Primer capítulo. Marco teórico: fundamentos conceptuales y contextuales

Señales: dominio acústico, eléctrico y digital

Un sistema está conformado por dos o más dispositivos que reciben, procesan y entregan señales de algún tipo: magnitudes variables en el tiempo que transmiten o transportan información. Para el caso particular de los sistemas de sonido existen dos tipos principales de señales: acústicas y eléctricas. (Miyara, 1999). La onda sonora, o energía acústica, consiste en la propagación de una perturbación en el aire, el cual está formado por una cantidad muy grande de pequeñas partículas o moléculas inicialmente en equilibrio dinámico. En reposo las moléculas están en movimiento pero homogéneamente repartidas, cuando se produce una perturbación aquellas partículas que se encuentren junto a la fuente serán empujadas, mientras que las que se encuentran más alejadas no serán alteradas. Esto implica que habrá más moléculas por centímetro cúbico cerca de la fuente que lejos de ella: como el aire comprimido tiende a descomprimirse, las partículas se desplazarán hacia la derecha, y comprimirán a su vez el aire que se encuentra próximo a ellas. El proceso se repite así en forma permanente —hasta que la amortiguación lo detenga del todo—, con lo cual la energía de la perturbación original se propaga alejándose de la fuente de dicha perturbación. (Miyara, 1999).

Un circuito eléctrico es un conjunto de componentes interconectados por medio de hilos conductores (cables), de tal modo que existe uno o más lazos cerrados por los que circula corriente eléctrica, que no es otra cosa que cargas eléctricas en movimiento. La intensidad de corriente eléctrica, o corriente, se define como la cantidad de carga eléctrica que circula por un conductor, y se mide en una unidad denominada *amper*. La otra magnitud circuital relevante es la tensión. Esta se mide utilizando como unidad el *volt (o voltio)*, razón por la cual a veces se le llama voltaje, y es la diferencia de tensión entre dos puntos de un circuito. La conversión entre una corriente y una tensión se realiza con un elemento llamado resistor, el cual tiene asociado un valor llamado resistencia que se expresa en unidades de *ohm* (Ω). (Miyara, 1999). Las variaciones de tensión tomadas por un micrófono son análogas a las variaciones de presión que éste capta del aire: hay una transducción del dominio acústico al eléctrico.

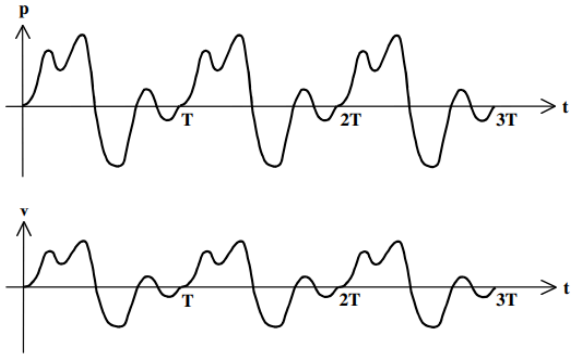

Figura 1. Presión sonora y su análoga, la tensión generada por un micrófono.
Nota. Relevado de *Acústica y Sistemas de Sonido* (p. 65), por Miyara, F (1999). UNR

El aprovechamiento de la electricidad fue uno de los más profundos desarrollos en la historia de los instrumentos musicales. (Katz, 2022). Antes de que el ser humano se apropiara de la misma la experiencia musical era efímera. La propagación en el aire del sonido brindaba una momento único que no podía volver a ser reproducido. El fonógrafo de Edison fue el primer transductor inventado, pasaba información del dominio acústico al mecánico: hizo para el sonido lo que los alfabetos fonéticos habían hecho para el habla incontables siglos antes. Luego, en la década de 1920, desde una nueva industria —el broadcasting— vinieron los micrófonos: invención que transformó la situación completamente al permitir que las grabaciones se manejaran de manera electrónica. Para el final de siglo ya contábamos con una tecnología radicalmente diferente, posible gracias al vertiginoso avance de la microelectrónica y su aplicación en dispositivos poderosos y complejos. Apareció una tercera manera de almacenar, transportar, generar y procesar audio: el audio digital. (Brooks, 2006).

La idea básica detrás del audio digital es la de representar el sonido por medio de números, con la gran ventaja de que “es mucho más fácil guardar un número que la magnitud que ese número representa” (Miyara, 1999) y permite nuevas técnicas de procesamiento de la señal sonora. Además, los algoritmos (métodos de cálculo) utilizados pueden implementarse en un dispositivo específico llamado procesador digital de señal (DSP) o bien en una computadora de propósito general. (Miyara, 1999).

Todos los sistemas digitales se basan en la numeración binaria. En esta se utilizan dos símbolos en un sistema posicional donde cada nueva cifra tiene un peso

sólo dos veces mayor que la anterior, a diferencia de las 10 veces mayor características de la numeración decimal. La razón por la cual se utilizan los números binarios es porque eléctricamente es muy fácil codificar los ceros y unos con nivel bajo y alto de tensión respectivamente.

La señal analógica se *muestrea*, es decir que se reemplaza por una serie de muestras tomadas a intervalos regulares (la frecuencia con la que se toman las muestras se denomina frecuencia de muestreo y el tiempo entre muestras período). Esta frecuencia determina la resolución en el eje horizontal o de tiempo y la resolución en el eje vertical o de amplitud es la profundidad de palabra o de *bits* con la cual se digitalice la señal.

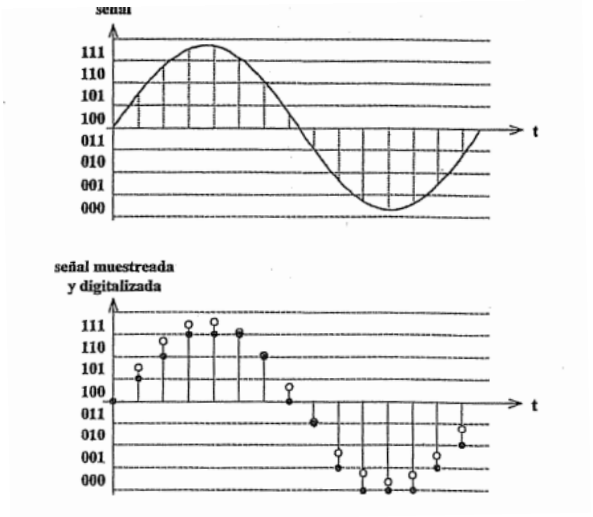

Figura 2. Señal continua y señal muestreada y digitalizada.

Nota. Relevado de *Acústica y Sistemas de Sonido* (p. 65), por Miyara, F (1999). UNR

Sistemas de sonido: dispositivos, usos y agentes

La cadena de un sistema de sonido comienza con una fuente sonora que emite energía acústica o eléctrica. Para el caso de la energía acústica se necesitará de un micrófono que reciba las variaciones de presión de la onda sonora y las convierta en variaciones de voltaje. Esa señal eléctrica se transporta hasta un preamplificador, externo o integrado a la mezcladora, que eleva su nivel para poder ser procesada con una buena relación señal/ruido. El procesamiento se puede dar de manera analógica o digital. En la primera la señal se mantiene en el dominio eléctrico y transporta la información en forma de una variación continua de voltaje

que evoluciona en el tiempo en forma análoga a la variable física. En el dominio digital la onda se codifica como una serie discreta de dígitos numéricos utilizando conversores analógico-digitales (ADC por sus siglas en inglés) y se transmite como variables eléctricas con dos niveles bien diferenciados que se alternan en el tiempo. (Miyara, 2004).

El procesamiento de las señal anteriormente mencionado es llevado a cabo por la consola de audio: dispositivo de la cadena encargado de “proporcionar la suma de las diversas señales eléctricas, cada una de ellas atenuada o amplificada con respecto a su nivel original en un factor ajustable por el operador.” Además están las funciones de procesamiento de dinámica, de espectro, paneo, temporales y las de “carácter administrativo” que facilitan el trabajo del operador en cuanto a ajustes, localización de errores, flexibilidad de conexionado, procesamiento o control de nivel por grupos, y demás. (Miyara, 1999).

Una vez procesada, la señal se envía al sistema de refuerzo sonoro: primero a los procesadores de sistema (crossover, alineamiento temporal, ecualización de recinto) y luego a las etapas de potencia, que elevan la señal a niveles capaces de excitar los altavoces. Estos transforman la energía eléctrica nuevamente en energía acústica, reconstruyendo el campo sonoro destinado al público si se trata del sistema de amplificación principal, que puede entenderse como la conjunción entre el PA (*public adress*) y las diferentes secciones llamadas *fields* que son colocados con una distribución espacial particular que busca cubrir el espacio.

Dentro de la industria del sonido en vivo existen distintos roles que cumplen los agentes presentes en un proyecto y en muchas de estas distribuciones podemos encontrar una superposición entre la técnica y la artística. El operador o sonidista es uno de los tres tipos de ingenieros de audio principales presentes en un show de sonido en vivo: el encargado del sistema principal, el encargado del sistema de monitoreo y el encargado técnico. El primero (el operador) es quien maneja la consola de audio principal y está encargado de realizar las mezclas que escucha el público, tomando decisiones tanto técnicas como creativas. Recibe las señales provenientes de toda fuente sonora que se desee amplificar y luego de procesarlas son ruteadas al sistema de parlantes que se esté utilizando. El monitorista es responsable de controlar lo que se escucha dentro del escenario: se encarga de las mezclas individuales de cada uno de los músicos. Los técnicos, a diferencia de los

dos anteriores, no operan los equipos durante el show, son quienes diseñan, montan y calibran el sistema.

La operación en vivo está atravesada por la instantaneidad. Las decisiones técnicas deben tomarse en tiempos extremadamente reducidos, bajo condiciones de presión y con interfaces cada vez más complejas. Estas problemáticas justifican la necesidad de herramientas y dispositivos auxiliares —como el desarrollado en esta investigación— que permitan optimizar tareas específicas del operador y mejorar la eficiencia en la toma de decisiones durante el show.

Protocolos de comunicación y entornos de programación

Protocolo MIDI

MIDI (*Musical Instruments Digital Interface*) es un protocolo de comunicación digital entre instrumentos musicales y dispositivos. En 1981, Dave Smith y Chet Wood proponen un protocolo de comunicación basado en una interfaz serie, el protocolo USI (Universal Sintetizer Interface). Se realizan algunas modificaciones y se publica *The Complete SCI MIDI*. Tras unos meses de experimentación, surge *MIDI Specification 1.0* donde se definen los mensajes y las características físicas y lógicas de la transmisión MIDI. (Melczarsky,2004). Hoy en día es un protocolo muy estandarizado y ampliamente utilizado. A través de él no se envía sonido sino eventos, por ejemplo cuando se pulsa una nota (Note On) o cuando se varía un control (Control Change). El protocolo está formado por mensajes que consisten en una cadena de bytes con algunos ya definidos. El MIDI también permite sincronizar y secuenciar instrumentos o almacenar una interpretación para su posterior edición y reproducción. (Melczarsky,2004).

MIDI es un protocolo digital: usa el sistema binario, donde el bit (unidad mínima de información) tiene 2 posibilidades, 0 y 1. Esto viene del control por voltaje apagado-encendido. Los bytes son conjuntos fijos de 8 bits donde tenemos 2^8 posibilidades, lo que nos da 256 combinaciones posibles pero de las cuales sólo utilizamos un rango de 128 valores (de 0 a 127 por ejemplo para los valores de nota) por byte. Algunos bits tienen usos específicos y están destinados a facilitar la

comunicación con la máquina. Todos los mensajes MIDI comienzan con un byte de estado, que indica el tipo de mensaje y número de canal si corresponde. Dependiendo del tipo de mensaje tendremos luego entre 0 y 2 byte de datos, que indican diferentes parámetros y valores según corresponda. Para distinguir entre un byte de estado y de datos, se utiliza el bit de la izquierda: si es un uno, se trata de un byte de estado y si es un cero corresponde a un byte de datos. Es por esta razón que el rango de posibilidades se reduce a 128: en vez de contar con 8 bits contamos con 7 porque uno se destina a indicar el tipo de byte y eso nos da 2^7 posibilidades, es decir 128 valores utilizables. Además, como las computadoras suelen trabajar con el sistema *zero-based* los datos están siempre comprendidos entre los valores decimales 0 y 127 (binarios 00000000 y 01111111). (Puig, 1997).

Algunos tipos de mensaje MIDI son el *Note on* que utiliza tres bytes (*Status byte* que indica el tipo de mensaje (en este caso nota), *Databyte 1* que indica el número de nota y *Databyte 2* que indica el valor de *velocity* o intensidad), el *Note off* (necesario para que una nota no siga sonando indefinidamente sino que se detenga cuando se levanta la tecla del controlador), el *Control Change (CC)* y el *Program Change (PC)*, en los cuales ahondaremos más en profundidad por ser de relevancia para el trabajo. El cambio de programa (o *Program Change*) se suele usar para elegir diferentes sonidos disponibles en un sintetizador y en las consolas se usan *presets* con configuraciones guardadas para poder recuperar escenas específicas. Este mensaje modifica el programa activo. Los mensajes de *Control Change* es particular porque engloba 128 posibilidades de control diferentes, donde cada uno de ellos afecta de distinta forma al sonido: dentro de los sintetizadores existen controles para modificar el volumen, la modulación, la reverberación, el delay, etc. Como indica Puig en su capítulo de *Especificación MIDI a fondo,* podemos analizar su estructura en el más bajo nivel: “el primer byte indica el tipo de control. De los 128 controles posibles, tan solo una pequeña parte está asignada, por lo que todavía quedan muchos por definir en un futuro. El segundo byte indica el valor de este control. La mayoría de controles utiliza la escala del 0 al 127, pero algunos funcionan únicamente de forma binaria (on/off).” (Puig, 1997).

Muchos sintetizadores implementan otros controles de forma no estándar. En muchos casos el efecto no se consigue con un solo control, sino mediante la combinación ordenada de varios mensajes de control determinados. Aunque esto suele ser bastante más complicado, de este modo se puede llegar a controlar al

máximo las posibilidades del sintetizador. Estos grupos de mensajes pueden ir desde el RPN y el NRPN a los mensajes Sysex (mensajes de sistema inclusivo). Como todo dispositivo MIDI posee algunas características peculiares, difíciles de incluir en un estándar, los fabricantes acordaron dejar un grupo de mensajes de formato libre, para uso particular de cada dispositivo. “Para lograr esta flexibilidad y privacidad, los mensajes de sistema exclusivo (Sysex) incluyen después del byte de status (que en este caso es siempre F0H ó 1111 0000 en binario), un byte con un código propio del fabricante (Roland, Yamaha, etc.) y otro específico del modelo. A continuación, el mensaje puede tener cualquier longitud, por lo que para indicar el fin del mensaje, se incluye el byte de status EOX (End of Exclusive) que vale F7H (1111 0111).” (Puig, 1997). Por otro lado, los mensajes del tipo RPN (Registered Parameter Number) constituyen parámetros oficiales, estandarizados por MIDI, con funciones específicas definidas por la norma y NRPN (Non-Registered Parameter Number) permiten controlar parámetros propietarios o específicos de un sintetizador o dispositivo, pero siguiendo un formato estándar dentro del MIDI, a diferencia del Sysex. Un mensaje NRPN se compone de cuatro bytes que trabajan en conjunto: los dos primeros definen el número de parámetro a través del CC 99 y el 98 y los dos últimos definen el valor transmitido a través del CC 6 y el 38. Por los CC 99 y 6 recibimos el byte más significativo (most significant byte), donde el peso de cada bit es de 128 y por los CC 98 y 38 encontramos el byte menos significativo (less significant byte) donde cada bit pesa 1. De este modo podemos contar con una profundidad de 14 bits, a diferencia de los 7 de los mensajes MIDI sencillos de un sólo byte, y tenemos por lo tanto de valores del 0 al 16.383. (MIDI Protocol, Allen & Heath).

Pure Data

En paralelo al desarrollo de la tecnología MIDI a finales de siglo XX también se comenzaron a desarrollar lenguajes de programación dedicados específicamente para trabajar con audio. Entre ellos Pure Data, un entorno gráfico de programación de código abierto diseñado para el procesamiento de audio, desarrollado en 1996 por Miller Puckette. Está basado en el funcionamiento de un lenguaje anterior, MAX, pero es más simple y portable: puede correr en cualquier computadora, celular o sistema embebido. A pesar de ser tan liviano ofrece una amplia variedad de paquetes de objetos externos y nativos que permiten generación de audio, control

de señal, manejo de sensores, conexión con dispositivos de entrada y de salida, dispositivos MIDI, como también procesamiento de imágenes y video. (Manual de Pure Data).

Pure Data permite a músicos, artistas, desarrolladores e investigadores crear un software de manera gráfica sin usar líneas de código. Las funciones del algoritmo son representadas con “cajas” (*boxes*) llamados objetos, cada uno con una tarea específica, que se posicionan en una ventana denominada *patch* o *canvas*. El flujo de señal entre los objetos se logra conectando visualmente “cables” o *patch cords*. Los objetos incluyen los nativos de PD y pueden incorporarse librerías de externos, compilados de C o C++, y abstracciones, subpatches cargados como objetos. (Página web oficial de Pure Data).

Cómo usar PD

Ni bien se abre el programa lo que primero que se visualiza es la consola, la ventana principal de PD, donde podemos imprimir mensajes y también visualizar información que el entorno nos brinda, principalmente para corrección de errores (*debuggear)*. Si creamos un archivo *.pd* se va a abrir una ventana distinta, en blanco, a la que llamamos *patch*. Allí podremos elegir entre el modo edición, en el que creamos el código y el modo performativo, usado para correr el programa: cambiamos entre un modo y el otro con ctrl + e, activando y desactivando el modo edición.

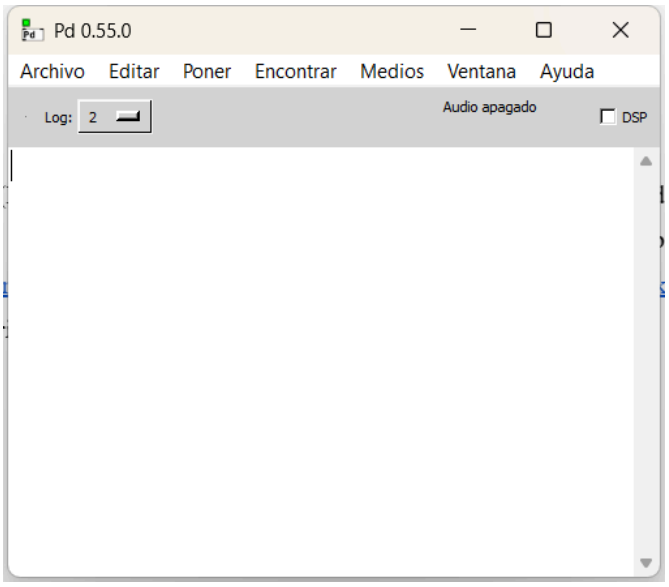

Figura 3. Consola de PD

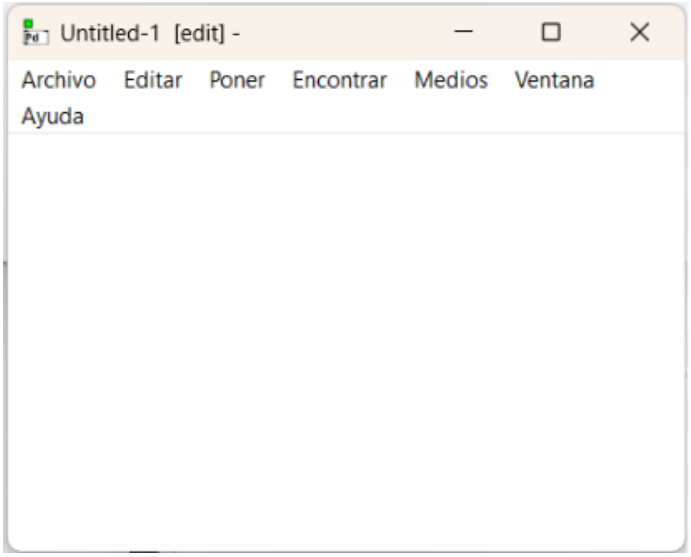

Figura 4. Patch vacío

Objetos

Un objeto en PD es un bloque funcional que realiza una tarea específica dentro del patch. Se puede crear un objeto yendo a la sección *poner* y seleccionando allí el tipo de objeto que se quiere crear o usar los atajos con ctrl + n.

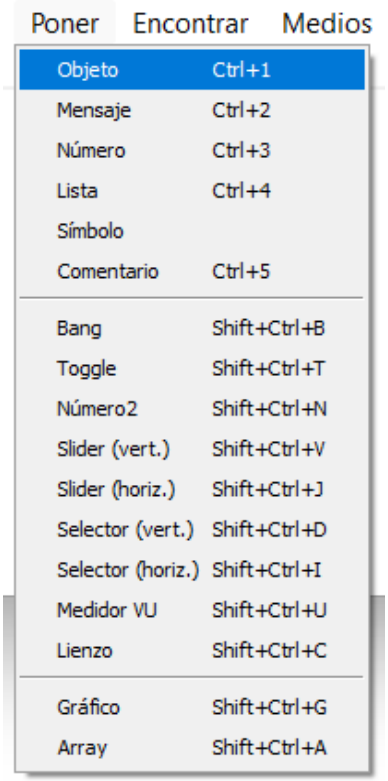

Figura 5. Lista de objetos

Figura 6. Objetos válidos y otros no conformados

Los objetos conformados se ven como una caja cerrada, los no conformados permanecen en línea punteada. Se puede observar que PD es sensible a la diferencia entre minúsculas y mayúsculas (case sensitive).

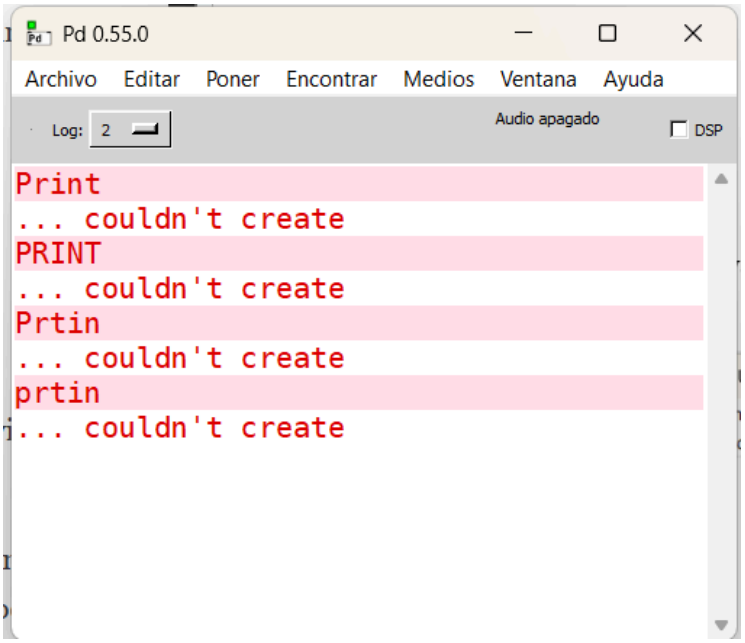

Figura 7. La consola nos muestra que los objetos mostrados anteriormente no fueron conformados porque no existen y por eso aparecen como errores

Tanto los objetos como las cajas de números y de texto tienen *inlets* (arriba) y *outlets* (abajo) por donde podemos interconectarlos: por el o los inlets *ingresa* información y por el o los *outlets* sale. El inlet izquierdo funciona siempre como *inlet*

caliente y el derecho como *inlet* frío: el frío se usa para ingresar o dejar guardado un valor (tanto números como símbolos) y el caliente, además de permitir también el ingreso de un valor, ejecuta al objeto.

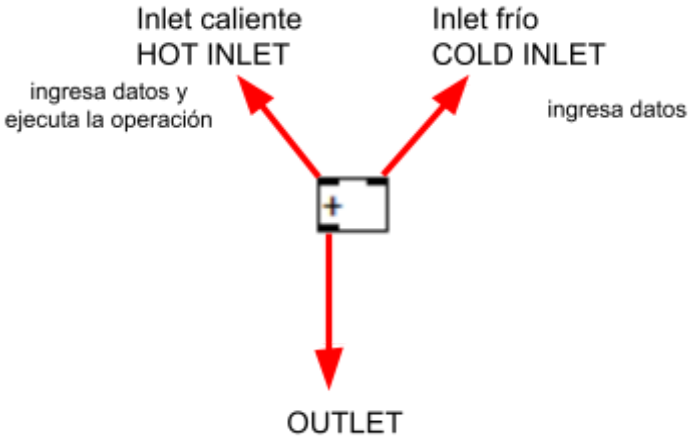

Figura 8. Tipos de inlet

Una forma de forzar a que ambos *inlets* detonen el cómputo o la operación es a través del objeto [trigger], por ejemplo en su formato: [t b f] (*trigger, bang, float*). Éste, cuando se ejecuta, devuelve primero el valor flotante (ingresado por el *inlet*) por el *outlet* derecho e inmediatamente después devuelve un *bang* por el *outlet* izquierdo. Este tipo de *trigger* es muy usado para forzar el output de algún objeto a través de su inlet izquierdo (siendo este el frío, el que ingresa un valor pero no ejecuta)

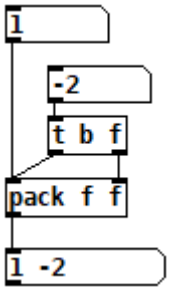

Figura 9. Trigger bang float.

A su vez, los objetos se conforman de átomos, separados por espacios. El primer átomo debe ser un símbolo que defina el tipo de objeto y el resto, que se conocen como argumentos, pueden ser símbolos o números que inicializan al objeto.

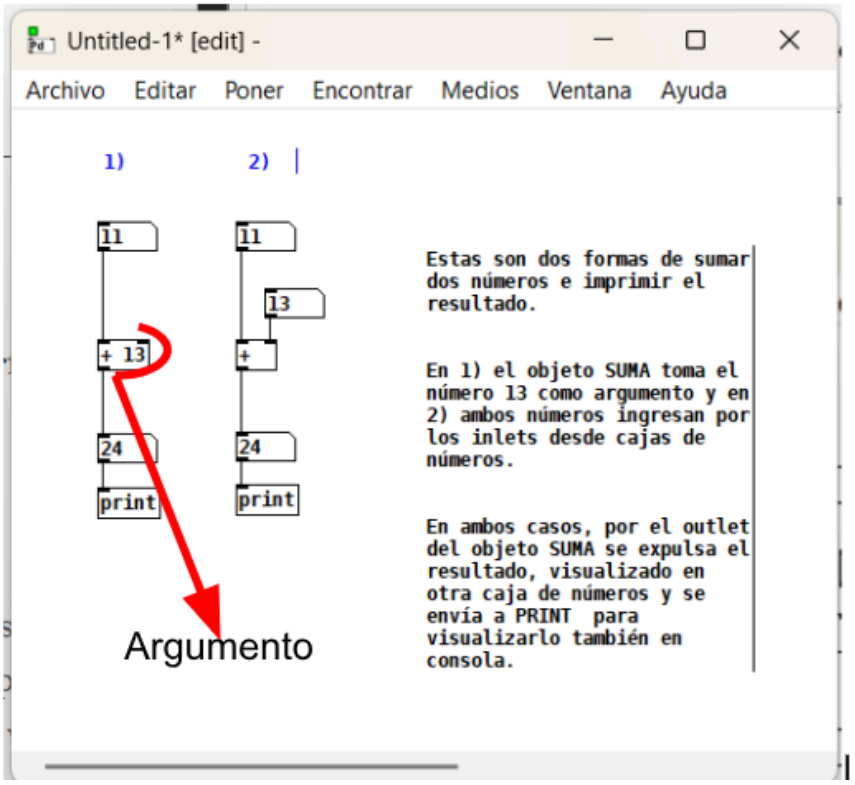

Figura 10. Ejemplo de patch muy sencillo que muestra el uso de argumentos

Para entender el funcionamiento de un objeto es muy útil entrar en su ayuda haciendo click derecho sobre el mismo. La ayuda es también un patch de PD, por lo que puedo copiar código de ahí mismo para pegar en mi patch. Las referencias son también de mucha ayuda, principalmente porque nos muestran qué puede ingresar por cada inlet, qué emite el objeto por el outlet y qué argumentos puedo usar.

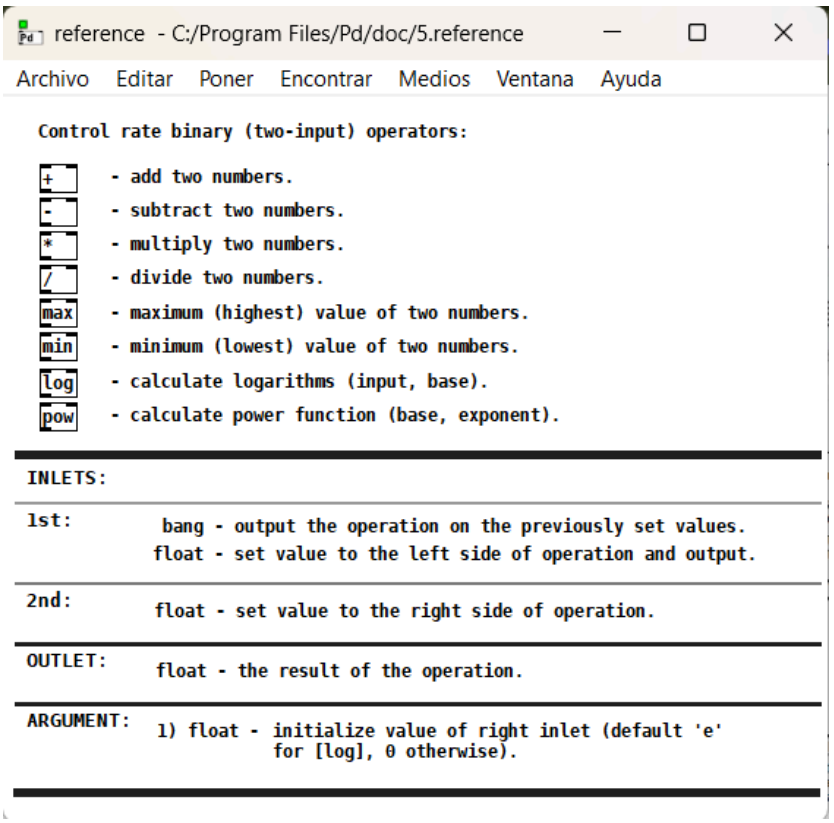

Figura 11. Se accede a las referencias clickeando [pd reference]. (Ubicado arriba a la derecha en la ayuda)

En Pure Data podemos categorizar a los objetos en dos grandes categorías: los objetos de control (*control objects*) y los objetos de señal (*signal objects*). Los primeros trabajan con datos: valores determinados y discretos y los segundos generan un flujo continuo de información (... aunque no del todo cierto porque estamos en el dominio digital). Las conexiones (las líneas entre las cajas) también son de tipo "control" o "señal". Visualmente son distintas: las conexiones de señal son más gruesas que las de control. El tipo de conexión depende de la salida de la que provenga.

El DSP (Digital Signal Processing) es un procesador de señales digitales que necesitamos para realizar cálculos matemáticos en tiempo real. Es una arquitectura especial pensada para poder computar de manera rápida y eficiente. En la esquina superior derecha de la ventana principal de Pd, se encuentra el interruptor DSP, que activa y desactiva el procesamiento de audio globalmente. El DSP está desactivado por defecto al abrir Pd. Al activarlo, Pd comienza a procesar muestras de audio en tiempo real para todos los *patches* abiertos (visibles o no). Es importante conocer el funcionamiento de los DSP si se quiere trabajar en computación orientada al audio y

por eso está incluído dentro de este marco. Sin embargo, siendo que el *patch* propuesto en este desarrollo no procesará señales para evitar problemas de latencia en la cadena de comunicación no es necesario que el DSP esté encendido.

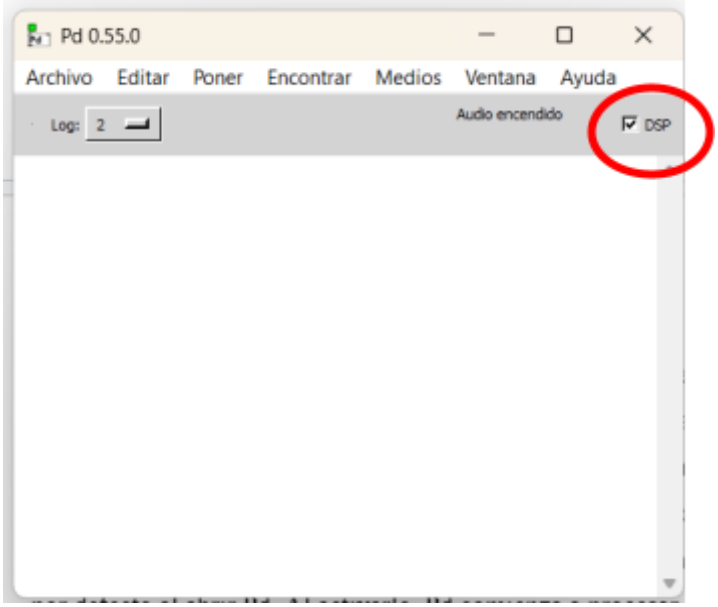

Figura 12. DSP encendido.

Para el envío de información de manera remota, "inalámbrica" o sin los "cables" que vemos cuando conectamos los objetos en PD se pueden utilizar los objetos [send] y [receive], o en su forma abreviada [s] y [r], donde el argumento debe ser el mismo en ambos para que logren enviar y recibir los mensajes respectivamente.

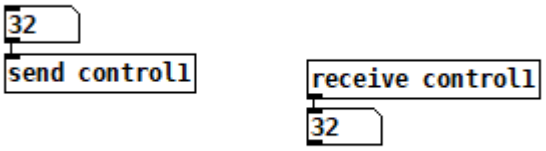

Figura 13. Mensajes de [send] y [receive] para enviar info de manera remota, en este caso de una caja de números a otra.

En este caso los argumentos no coinciden: no se recibe el mensaje.

Figura 14. [send] y [receive] abreviados. Funcionan de la misma forma.

Otra manera de enviar mensajes de manera remota es indicando a un objeto desde sus propiedades un símbolo de envío/recepción. Notar que cuando se utiliza esta forma el outlet (si hacemos envío remoto) o el inlet (sí hacemos recepción remota) desaparecen. No podemos conectar “cables” porque estamos forzando a que las conexiones sean “inalámbricas”.

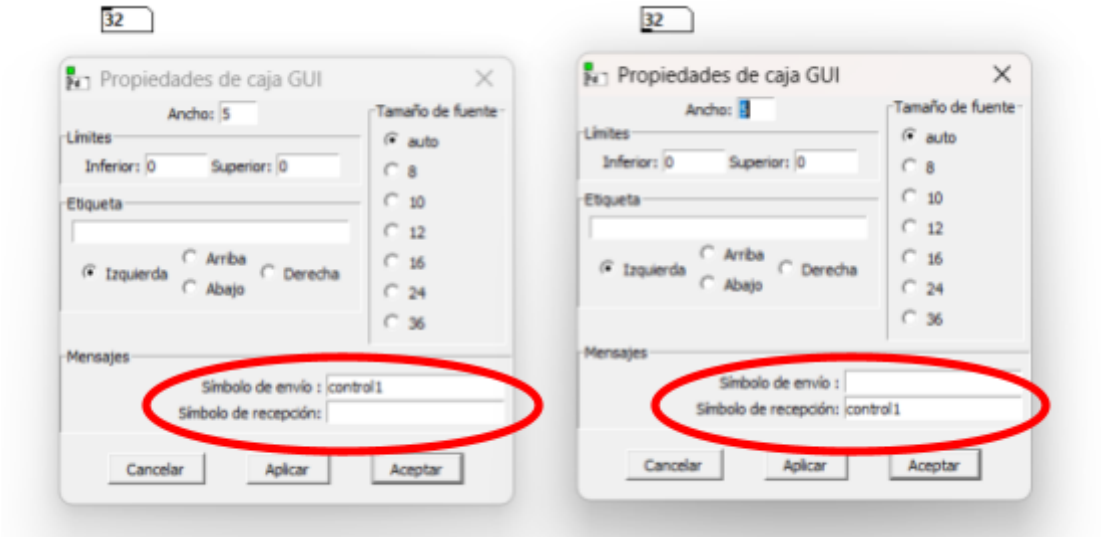

Figura 15. Propiedades de una caja de mensajes. Control de símbolos de envío y de recepción remotos.

Una tercera manera de realizar envíos remotos es a través de cajas de mensajes, donde el punto y coma (;) hace que la información de la caja de mensajes salga fuera de la misma. El primer átomo del mensaje será el remitente y lo que le sigue es el mensaje que será enviado.

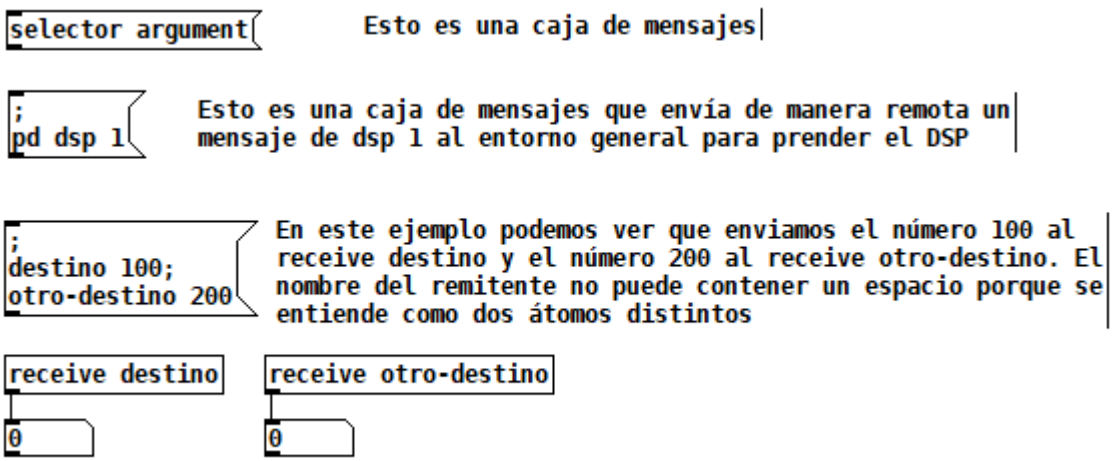

Figura 16. Envío remoto a través de mensajes

Por último, las variables son un espacio de memoria con un nombre específico donde se almacena un valor. Ese valor puede cambiar durante la ejecución de un programa. En *Pure Data* formamos con un signo de dólar y un número (como “$1” o “$2”) una variable que se sustituye con valores suministrados. Hacen referencia a los argumentos del mensaje que llega (y no están definidos si envía un mensaje "*bang*" o si hace clic en el cuadro de mensaje para activarlo). Las variables con $ pueden ser tanto números como símbolos, dependiendo del tipo de dato que se recibe.

Librerías externas

Pure Data ofrece una amplia variedad de paquetes externos que aumenta sus capacidades, muchos de ellos se enfocan en el procesamiento de audio y señales pero también hay para implementar el uso y procesamiento de gráficos y demás. La única librería de *externals* necesaria para correr el programa desarrollado en esta investigación es *cyclone*.

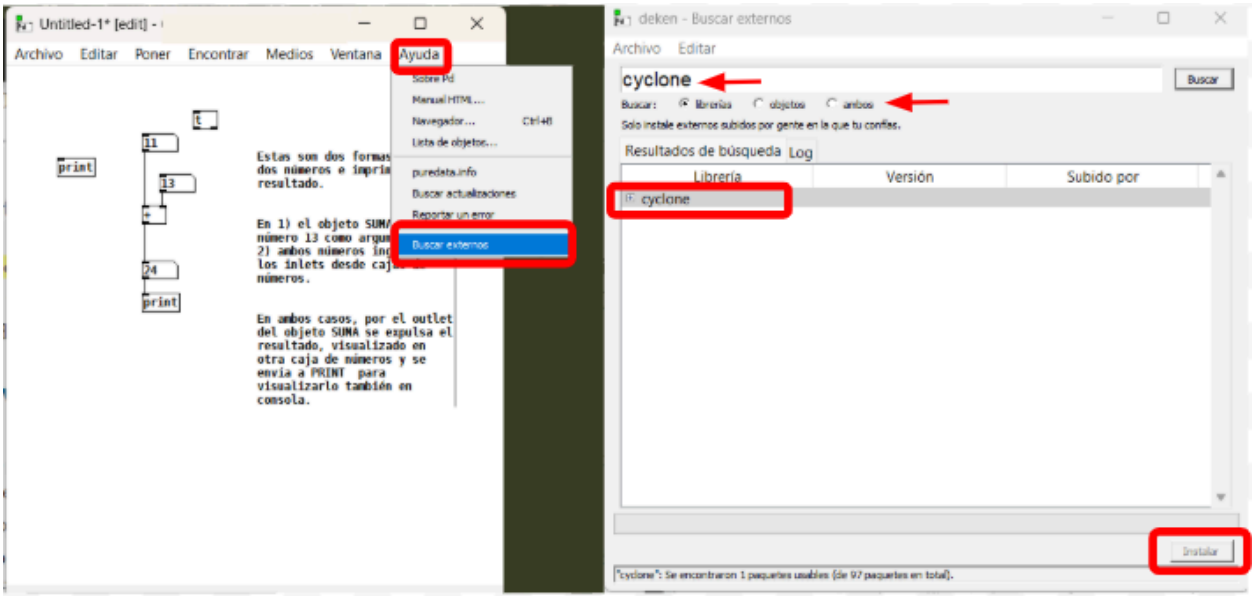

Figura 17. Para descargar librerías de externos desde el *deken* (directamente desde PD) se debe ir a *Ayuda→Buscar externos*. También puedo descargarlos en la PC de manera independiente.

Posteriormente a haber descargado la librería debo declararla, con el objeto [*declare]*.

```
declare -lib cyclone
```

Figura 18. Declaración de la librería externa utilizada

Puede ser que también se necesite declarar la ruta o dirección donde se encuentra ubicada la librería dentro del disco en Archivo→Preferencias→Editar preferencias.

Abstracciones y *subpatches*

Pd ofrece dos mecanismos para hacer subpatches: los llamados "*one-off subpatches*" y las "abstracciones". En ambos casos el *subpatch* aparece como una caja de objeto en el patch principal, la diferencia radica en que el *subpatch one-off* se guarda como parte del *patch* madre/principal, en un sólo archivo y la abstracción es un archivo independiente que puede ser reutilizado en varios proyectos siempre y cuando se encuentre alojado en la misma carpeta que el *patch* madre.

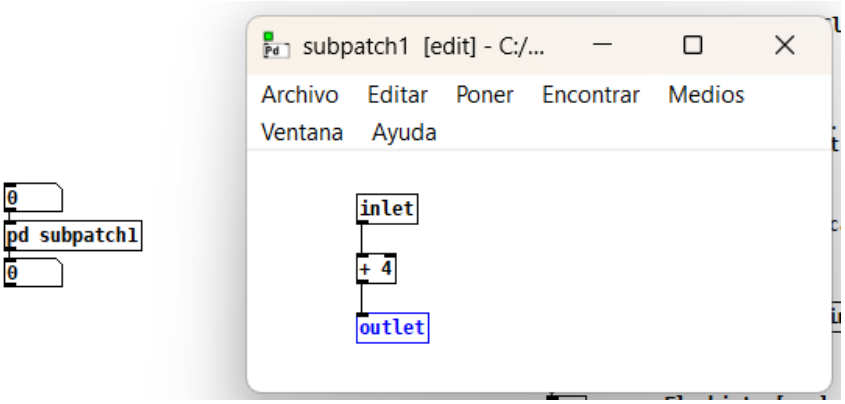

Figura 19. Ejemplo de subpatch con un (1) inlet y un (1) outlet.

Trabajando por instancias

Una opción que nos ofrece PD, que facilita el trabajo con copias y posibilita escalar las ideas a un tamaño y alcance mucho mayor, es la de trabajar con el objeto [clone]. Éste objeto es utilizado para automáticamente crear y manejar múltiples copias de una abstracción. Recibe como argumento el nombre de la abstracción, que debe estar alojada en la misma carpeta que el *patch* madre, y un número mayor a cero que representa la cantidad de instancias que se quieren inicializar. Los *inlets* y *outlets* del objeto [clone] corresponden a aquellos que existan dentro de la abstracción y pueden ser tanto de control como de señal.

La manera de establecer comunicación con las diferentes instancias creadas puede ir desde mensajes que ingresan por el primer *inlet* de [clone], cuyos métodos aceptados son: [resize(para modificar el número de instancias, [vis(para abrir o cerrar una instancia, [next(para enviar un mensaje a la instancia siguiente, [this(para enviar un mensaje a la instancia actual, [set(para setear el contador de instancias y [all(para enviar un mensaje a todas las instancias a la vez. Otra forma de comunicación con la instancia dentro de [clone] es de manera remota: se pueden tener [send] y [receive] por fuera de la abstracción que se comuniquen con la misma. Incluso, las instancias tienen un auto nomenclado que permite enviar un mensaje a una instancia particular conociendo su valor de $1 si se quiere saber el número de instancia o $0 como número único asignado a cada canvas.

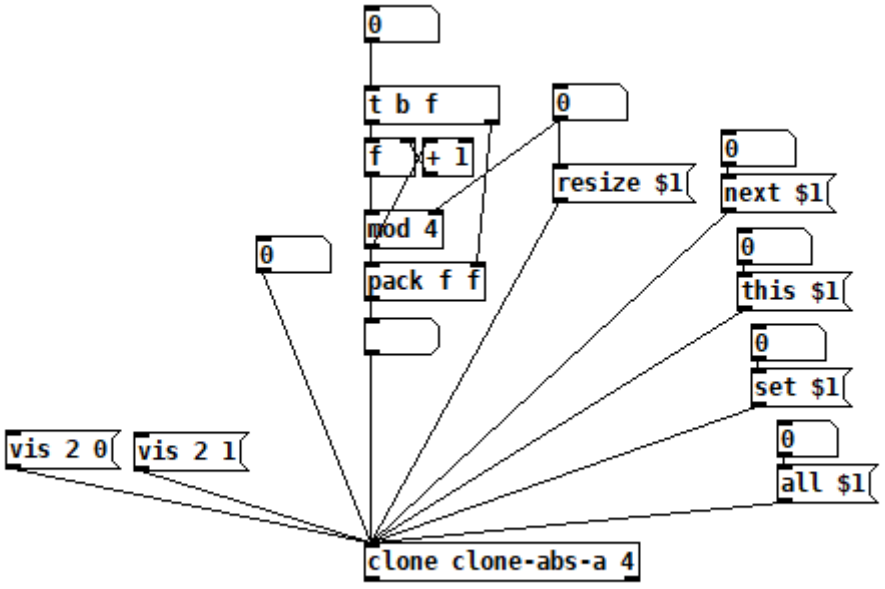

Figura 20. Objeto [clone] y sus métodos

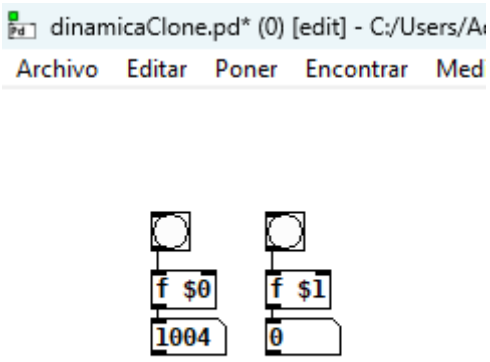

Figura 21. Instancia de clone: ID = 1004 y número de instancia = 0

Programación dinámica

A lo que llamamos programación dinámica en *Pure Data* puede ser encontrado en los foros como *dynamic patching.* El *modus operandi* de esta increíble y muy útil capacidad dentro del entorno es a través de mensajes remotos: se debe utilizar el punto y coma (;) para que el mensaje sepa que debe ser enviado a un remitente, el cual en este caso debe ser el nombre de un *subpatch* y a continuación se crea un mensaje que genera un objeto o que conecta dos objetos.

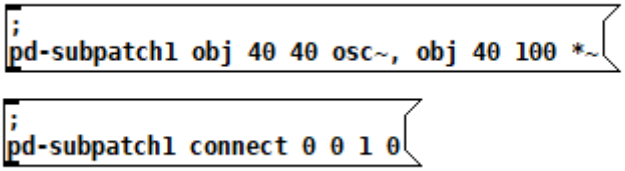

Figura 22. El primer mensaje crea un oscilador y un multiplicador, el segundo los conecta.

Para crear un objeto se utiliza el argumento *obj,* luego la posición en x y en y donde se graficará y seguido a esto el nombre del objeto. En el ejemplo se puede ver que se pueden separar dos mensajes con una coma (,) y en una misma caja de mensajes crear dos objetos(se sigue manteniendo el orden de creación). Para conectar dos objetos se utiliza el argumento *connect* y luego el índice de creación del objeto que se quiere poner primero en la cadena, seguido del número de *outlet* a utilizar. Seguido a esto se escribe el índice de creación del segundo objeto y su número de *inlet.*

Segundo capítulo
Trazabilidad del trabajo de campo

Segundo capítulo. Trazabilidad del trabajo de campo

Sobre la importancia de la bitácora en investigaciones de desarrollo aplicado

En esta investigación de desarrollo aplicado, la bitácora como registro escrito de las diferentes instancias del desarrollo constituyó una herramienta metodológica central que permite registrar de manera sistemática el proceso heurístico y técnico que conduce al prototipo final. (Polya, 1989). Este registro del proceso da cuenta de que el conocimiento emergió de las iteraciones, de la resolución de problemas y la reformulación constante de hipótesis de trabajo: no fue un proceso lineal. La bitácora documentó este proceso en su totalidad y otorgó trazabilidad a cada prototipo, lo cual habilita a futuro la reproducibilidad del desarrollo y ofrece transparencia respecto de los criterios técnicos y conceptuales adoptados. Asimismo, la bitácora operó como un dispositivo de externalización del pensamiento, mediante el cual se registró la evolución de los razonamientos, alternativas descartadas, problemas detectados y justificaciones metodológicas.

Este proceso de explicitación mejoró la consistencia interna del proyecto y transformó un proceso singular de resolución de problemas en conocimiento comunicable, al permitir el acceso no solo a la solución implementada, sino también a las estrategias que la hicieron posible. De este modo, el registro detallado del proceso contribuye a la construcción de una memoria técnica colectiva, facilitando la transferencia, el análisis crítico y la reutilización del conocimiento generado. A su vez, cabe recalcar que en la práctica profesional de programación es de vital importancia trabajar con repositorios, una práctica que se intentó copiar de este modo, guardando los distintos *patches* como archivos separados para poder volver sobre los mismos sin sobreescribirlos y fuertemente documentados para mayor claridad.

A continuación se describirán las diferentes facetas que llevaron al desarrollo a su etapa final, se incluirán capturas de pantalla del *patch* hecho en *Pure Data* acompañadas de descripciones generales de la instancia particular, qué se buscaba en ese momento resolver y una reflexión de si se logró o no, problemas que aparecieron y posibles soluciones o ideas para la próxima fase.

Primer prototipo

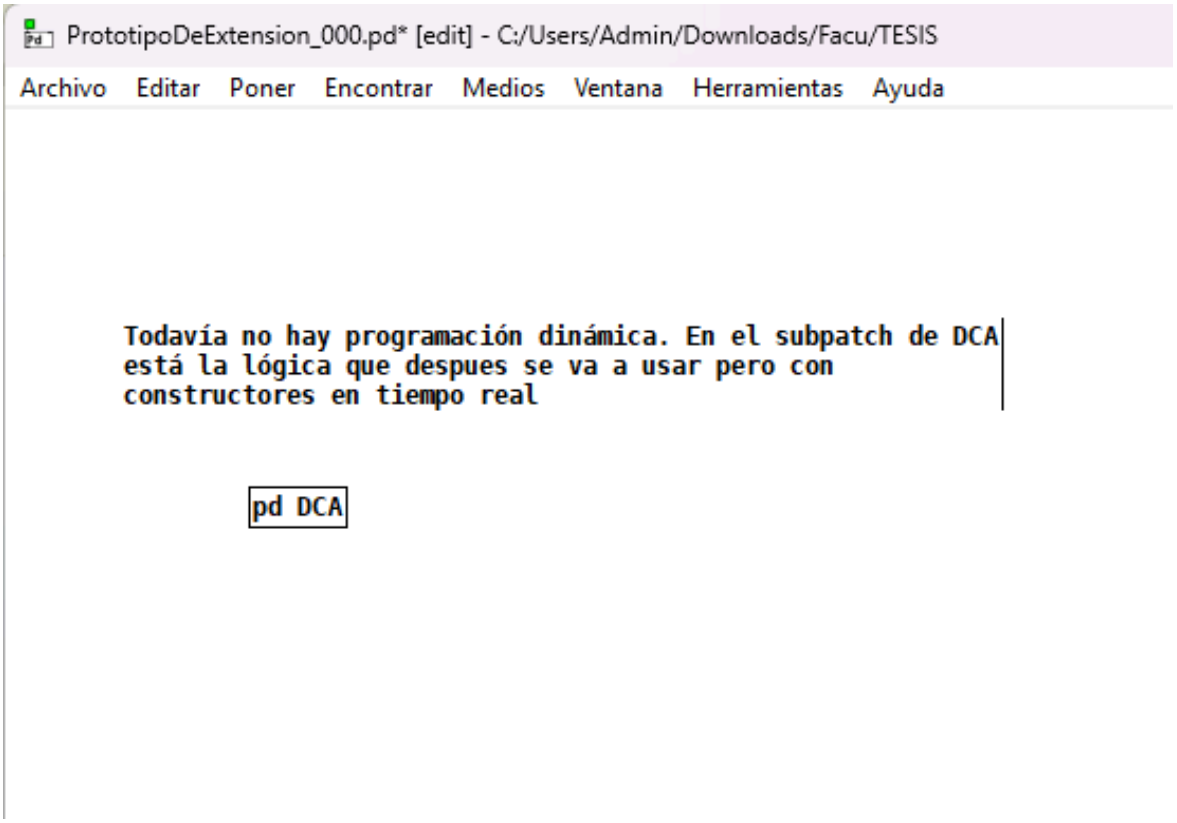

Figura 23. Patch de *Pure Data: PrototipoDeExtension_000*. *Patch* madre

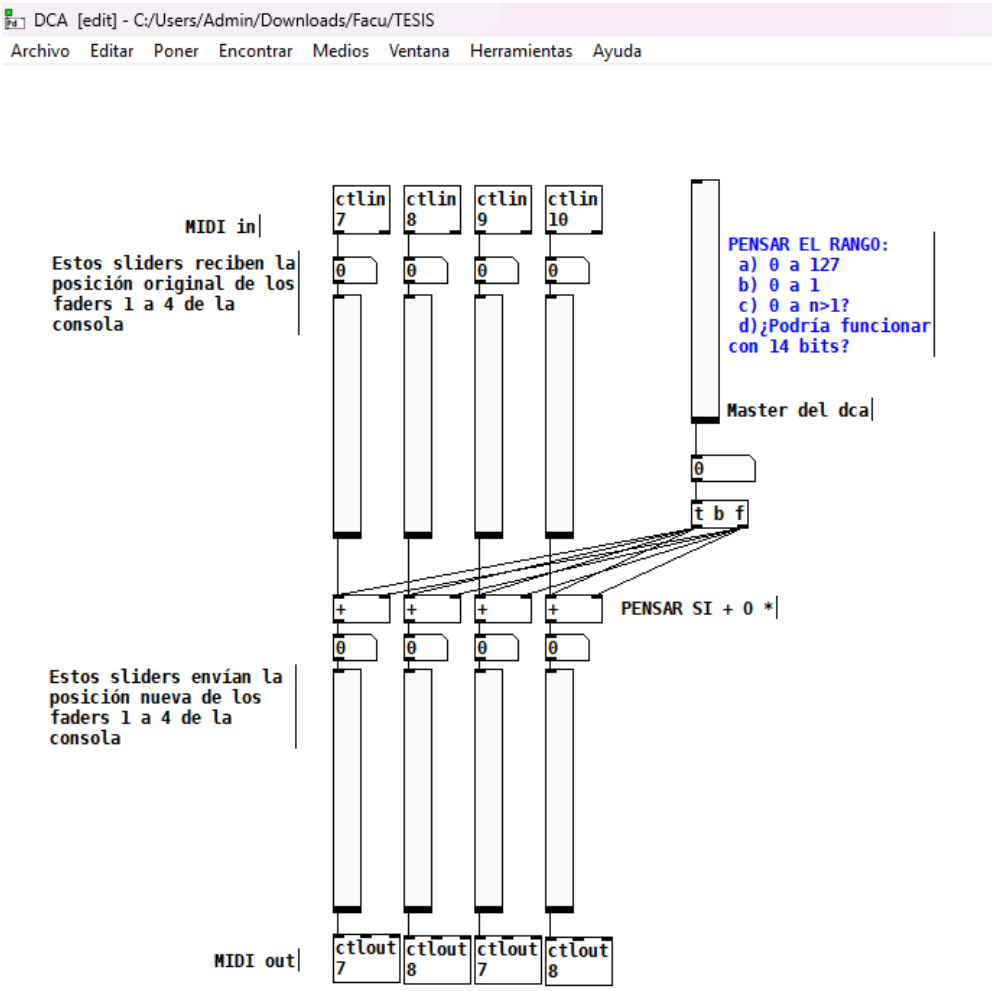

Figura 24. Patch de *Pure Data: PrototipoDeExtension_000*. Subpatch *DCA*. Foto 1

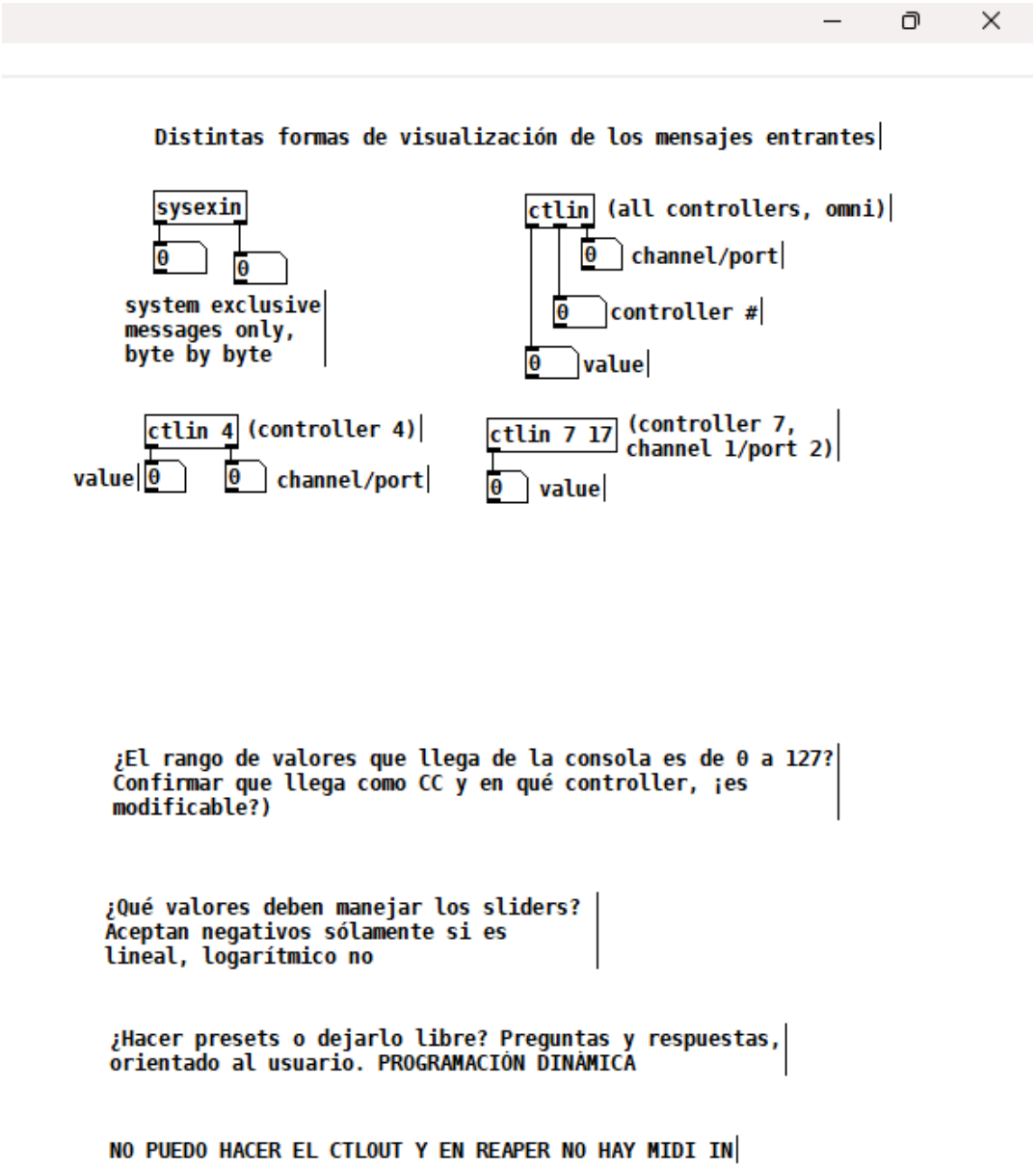

Figura 25. Patch de *Pure Data: PrototipoDeExtension_000*. Subpatch *DCA*. Foto 2

En esta instancia se ve reflejada la primera aproximación a la problemática. Se buscó compactar la idea general en el concepto más sencillo: programar un DCA virtual donde haya ingreso de información MIDI, visualización del parámetro ingresado, agrupamiento de canales, procesamiento del nuevo valor por medio de un *slider* maestro de grupo, visualización de la nueva posición una vez realizado el cómputo y la salida de información MIDI. Se constituyó así una base sólida del funcionamiento de los DCA para luego poder clonar y realizar de manera dinámica esta misma estructura. Todavía no se tenía el conocimiento exacto de qué iba a recibir el entorno desde la mesa de mezcla, es decir en qué *CC Controller* se iba a

recibir el cambio de posición de cada *fader*, por lo que se optó por simplemente escribir [ctlin n+1] y [ctlout n+1] empezando por n=0 para poder comunicarse con cada *channel strip* de la consola de manera independiente.

Primera prueba

Para testear el funcionamiento de esta etapa del proceso se comunicó a *Pure Data* con el *DAW* de licencia libre *Reaper* a través del cable virtual también de licencia libre *LoopMidi* pero el protocolo MIDI es en este caso mucho más amigable y conocido que en el caso posterior donde ya se incluye la consola. Aunque esta prueba fue fructífera y se logró la comunicación entre *softwares* y los presupuestos cambios en los *faders* esta prueba terminó resultando un impedimento para la investigación. Fue un problema porque se desestimó el tiempo de trabajo que iba a llevar la interpretación del mensaje real *MIDI IN* de la consola una vez conectada con la pc, en comparación con la emulación en la *Digital Audio Workstation.* El haber probado en *Reaper* fue una buena manera de constatar que el código era correcto y que las configuraciones de preferencias eran las adecuadas, además de que era la única opción viable mientras no se pudiera disponer de una consola de audio digital para hacer la prueba pero no debería haber sido considerada suficiente como para retrasar la prueba real con la mesa de mezcla SQ6 de Allen & Heath.

Segundo prototipo

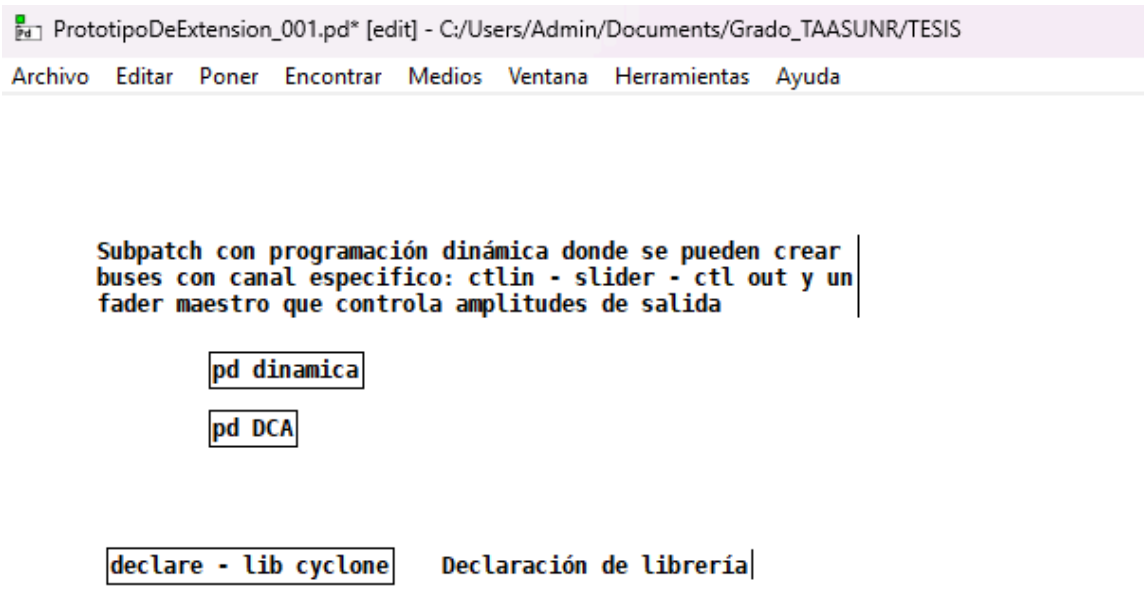

Figura 26. Patch de *Pure Data: PrototipoDeExtension_001*. *Patch* madre.

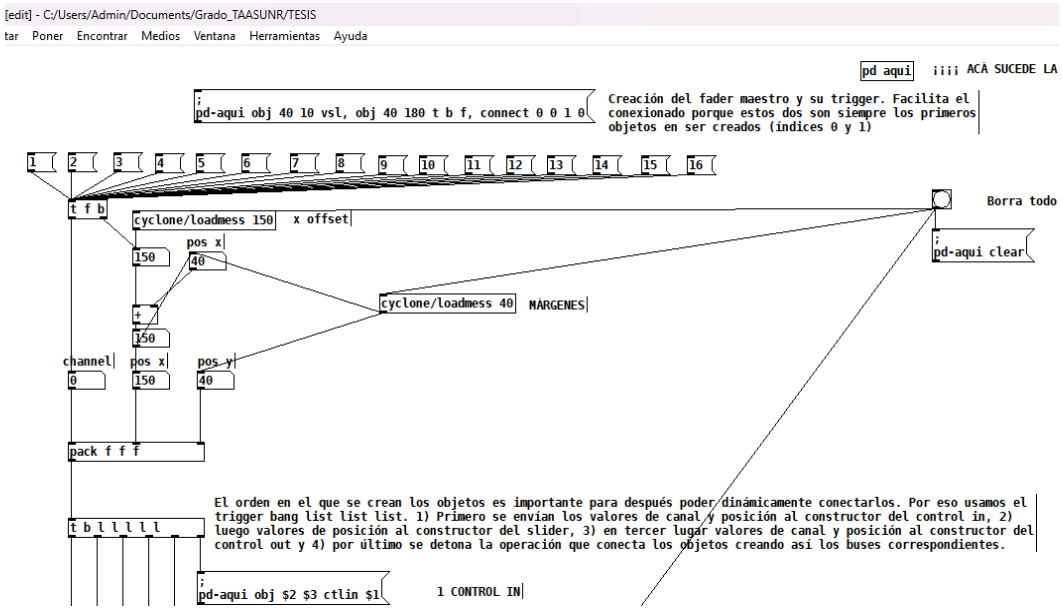

Figura 27. Patch de *Pure Data: PrototipoDeExtension_001*. Subpatch *dinamica*. Foto 1

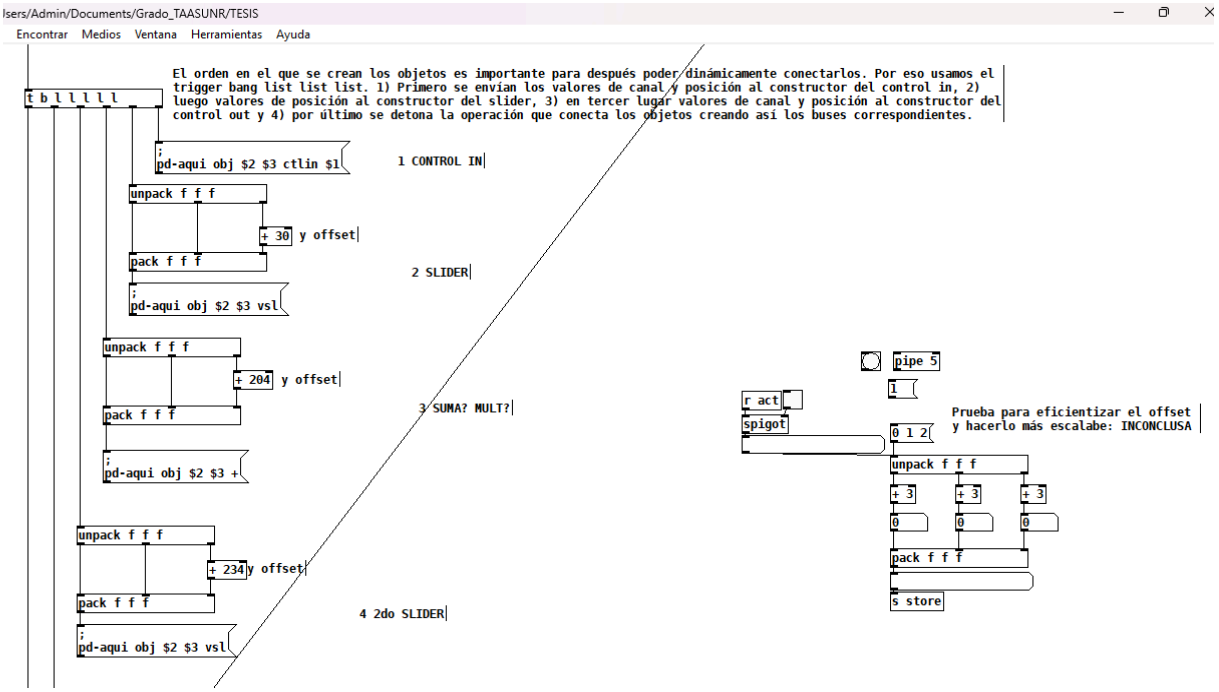

Figura 28. Patch de *Pure Data: PrototipoDeExtension_001*. Subpatch *dinamica*. Foto 2

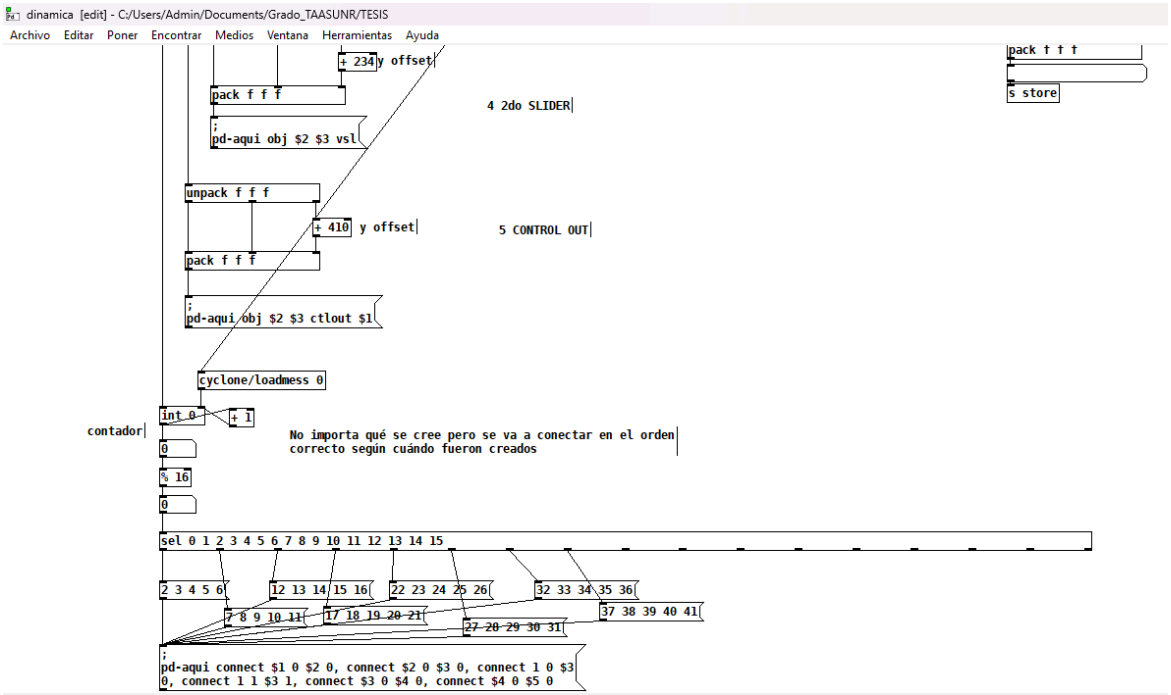

Figura 29. Patch de *Pure Data: PrototipoDeExtension_001*. Subpatch *dinamica*. Foto 3

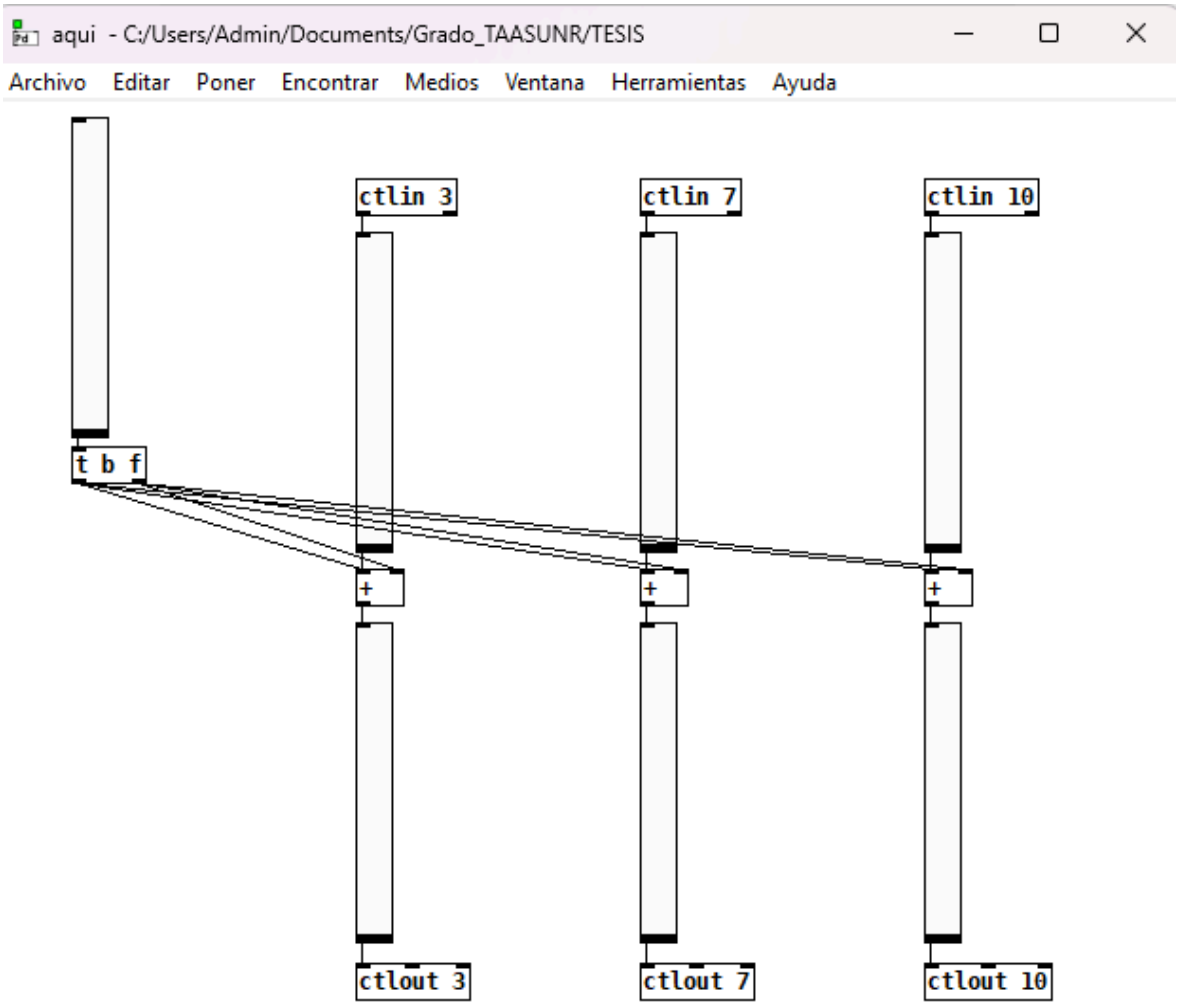

Figura 30. Patch de *Pure Data: PrototipoDeExtension_001*. Subpatch *aqui*

En este momento de la investigación se decidió avanzar sobre la flexibilidad y personalización del producto, por lo cual se empezó a buscar información acerca de la programación dinámica en PD. Se hizo un algoritmo sencillo para controlar el orden de creación de los objetos para su correcto conexionado y corrimiento u *offset* en los ejes horizontal y vertical para que éste sea legible. El resultado fue muy satisfactorio y cumplió con las expectativas de la investigación. Dentro del *subpatch* "*dinamica"* se puede hacer *click* en la primer caja de mensajes que crea el fader maestro y el [t b f] necesario para forzar ambos *inlets* de la suma a funcionar como *inlets calientes.* A través de cajas de números individuales se puede elegir qué canales se desean estén presentes en el grupo y se crean los *buses* con la lógica acarreada desde el *patch* anterior. Gracias a un *trigger bang list list list list list* ([t b l l l l l]) se desencadena la creación de objetos en el orden correspondiente pensado para después poder conectarlos según índices de creación. Los mensajes de

[;pd-aqui obj …(se comunican con el subpatch “*aqui”* y le indican que cree el objeto indicado como argumento, a cada uno de esos mensajes remotos les llega la lista con tres flotantes que corresponden de izquierda a derecha a número de canal, margen en *x* y margen en *y.* Los átomos de la lista son interpretados por cada mensaje como tres variables: $1, $2, $3. Para el caso particular del *offset* en el eje vertical se decidió desempaquetar la lista con un [unpack] ,sumarle a éste un valor que permitiese que los objetos no se superpusieran y volver a formar la lista con un [pack].

Se puede ver en la foto 2 del *subpatch dinamica* que se intentó dar con la forma de automatizar este corrimiento para poder escalarlo y no tener que estar seteando el valor de *y* a mano. Sin embargo, este experimento fracasó. El problema detectado que impidió poder realizar esta porción de código como se esperaba radica en que el objeto GUI de lista tiene un sólo *inlet*, el cual cada vez que recibe algo detona la lista por su *output* y no permite almacenar un valor sin ver el cambio reflejado inmediatamente. Esto es una prueba fehaciente de que los intentos inconclusos también generan conocimiento si se realiza una aproximación a ellos inteligente y con una honesta detección de las limitaciones encontradas.

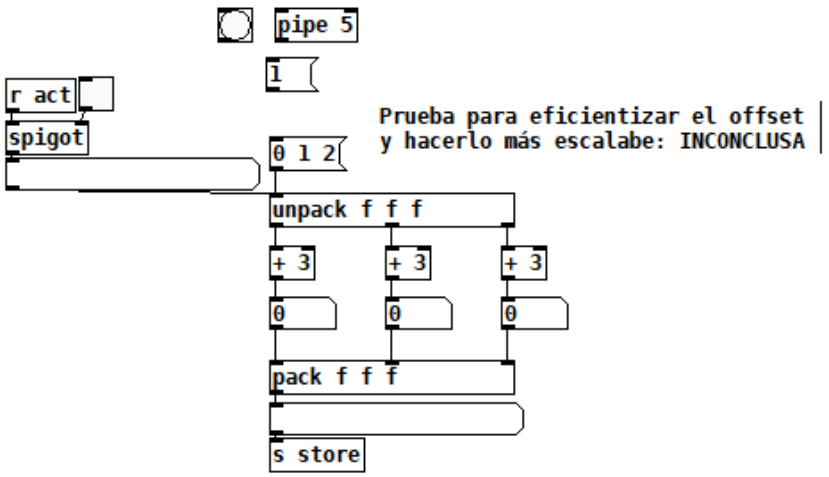

Figura 31. Patch de *Pure Data: PrototipoDeExtension_001*. Subpatch *dinamica*. Foto 2: acercada

Como el *subpatch* de programación dinámica se realizó en base a la plantilla pensada anteriormente en el *subpatch* DCA se trabajó siempre con la pantalla dividida, programando de un lado los mensajes remotos para crear y conectar objetos mientras se tenía la referencia en la otra mitad de la pantalla de cómo debía

ser el código final.

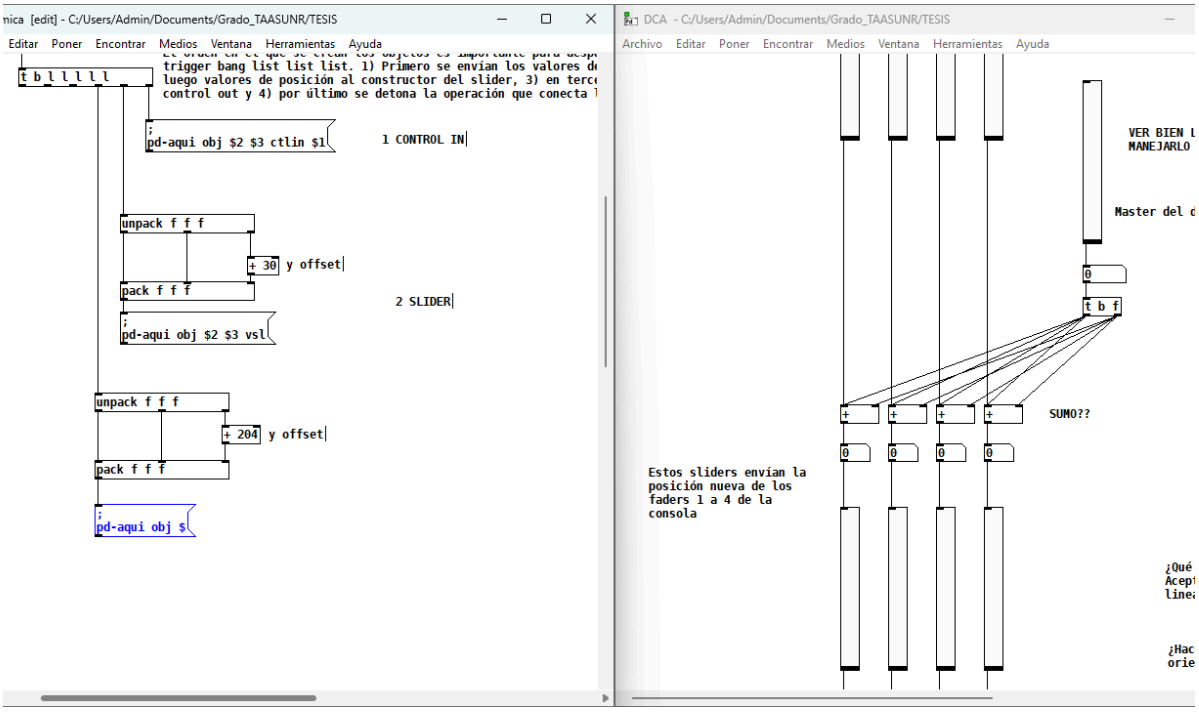

Figura 32. Vista de pantalla dividida

Segunda prueba

En este momento preciso del proceso encontramos un error en los presupuestos técnicos del protocolo MIDI. Se realizó, ahora sí, una primera prueba con el *patch* funcionando para probar la comunicación con la mesa de mezcla. Para esto se aprovecharon las instalaciones de la Escuela de Música, de la Universidad Nacional de Rosario: se trabajó en el EMEA, Estudio de Música Electroacústica, con la consola allí presente de la marca Allen & Heath modelo SQ6. Se conectó a la misma con la computadora mediante un cable USB A/b usando la conexión formato A en la pc y el b para la consola. No hizo falta el uso de una placa de audio externa ya que no se necesitó el uso de un cable MIDI de cinco pines y el USB, más ampliamente utilizado, puede ir directamente a un puerto USB de la computadora sin necesidad de un *hardware* externo.

Lo que se esperaba era recibir la posición de los faders por canales separados: se estimaba que cada uno de manera individual iba a enviar el valor de posición a través de canales separados o por valor de control diferente. Lo que se vio en esta segunda prueba con la consola fue que todos los *faders* transmitían la información por el mismo canal (1) y por el mismo controlador (38). Para evitar

cualquier problema derivado del código y no de la comunicación se decidió dejar de lado el *patch* en el que se estaba trabajando y realizar uno aparte a modo de visualizador, lo suficientemente sencillo como para descartar problemas en la arquitectura programada. Así se descubrió, bastante tiempo después, que lo que parecía un sólo número recibido eran en realidad cuatro. Se agregaron varios objetos [print], primero de manera individual lo cual ayudó a descifrar que los mensajes MIDI de un mismo *fader* estaban llegando por más de un controlador (99, 98, 6 y 38: siendo el último el que originalmente se veía en la caja de números y se pensaba ser el único). Luego se optó por empaquetar el controlador con el valor recibido para poder visualizar correctamente cuáles eran los pares de *Controller-Value* que estaban llegando al mismo tiempo.

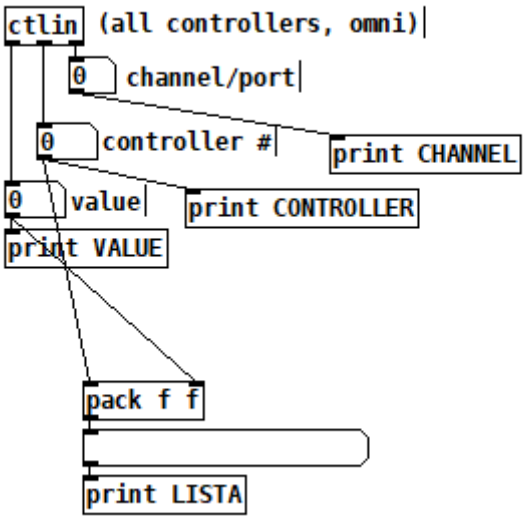

Figura 33. Patch de *Pure Data: Visualización.* Foto 1

Una vez realizado este proceso se tomó nota de lo que la consola devolvía en relación a distintos valores estratégicos del *fader* de la consola, mostrado a continuación. Con esta información actualizada se volvió a abrir el marco teórico, profundizando en el protocolo MIDI y las distintas opciones de comunicación entre dispositivos más allá de los mensajes simples convencionales. Se suponía que se estaba recibiendo un paquete de cuatro mensajes de *Control Change* por cada valor recibido y por lo tanto se buscó información de mensajes MIDI de mayor profundidad que utilizan dos mensajes para aprovechar el doble de bits disponibles.

Fue un gran desafío tener que modificar los presupuestos establecidos en la primera etapa del proyecto ya en un estado avanzado de la investigación.

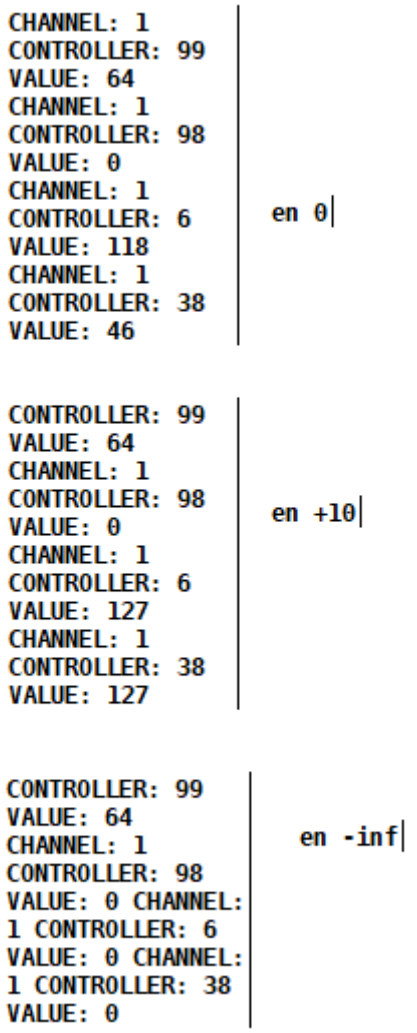

Figura 34. Patch de *Pure Data: Visualización.* Foto 2

Para poder continuar con el plan se necesitaba ir más seguido al EMEA de lo que se esperaba para poder ir constatando ideas con la consola. Como no era posible acudir rutinariamente se siguió investigando cuál podría ser el verdadero método de comunicación de manera teórica, buscando en artículos y libros sobre MIDI pero también en blogs informales donde programadores que utilizan el entorno y que tuvieron problemas similares obtienen respuestas de otros usuarios. Se encontró esto sumamente útil para poder comparar la situación con otras similares y analizar cómo problemáticas similares fueron solventadas. A la par de esta investigación más abstracta y alejada de la práctica se decidió continuar con la

siguiente fase del código en un tercer prototipo: trabajar con instancias de clonación para poder escalar el programa a múltiples grupos de manera eficiente.

Tercer prototipo

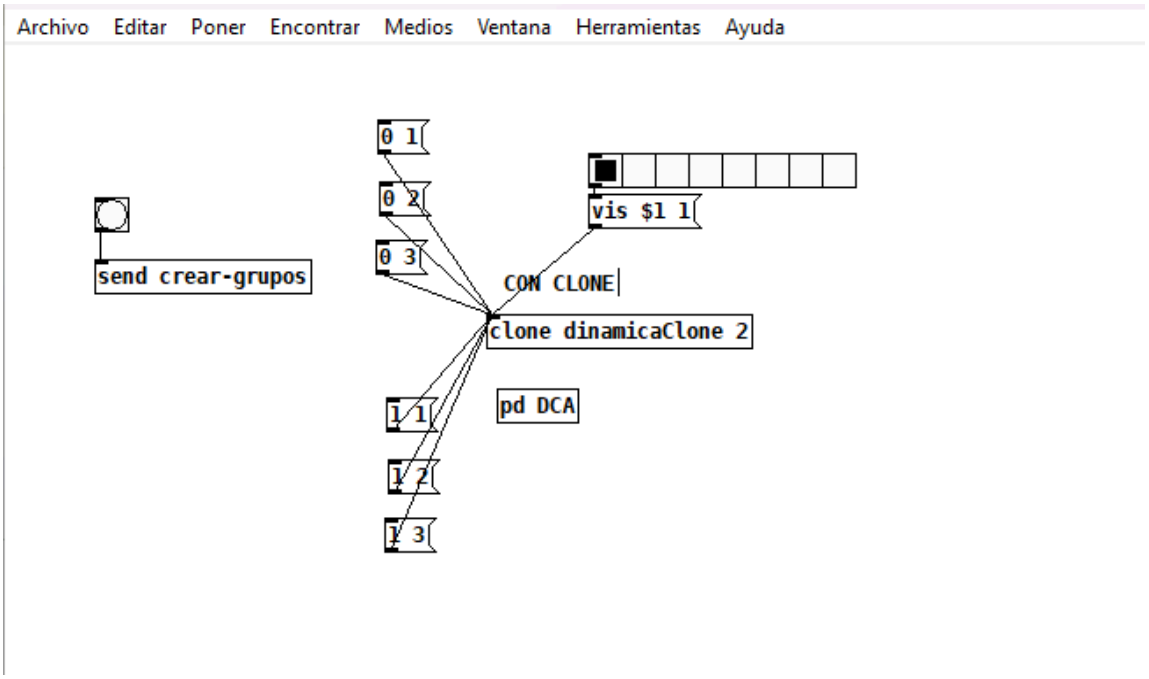

Figura 35. Patch de *Pure Data: PrototipoDeExtension_002*. *Patch* madre.

La gran diferencia de este *patch* con los anteriores es la escalabilidad. Se utilizó la programación por instancias con el objeto [clone] para poder trabajar con múltiples grupos a partir de la plantilla base. Para definir los canales de cada grupo en este prototipo se debe enviar mensajes al objeto clonador del tipo [x y(, siendo *y* el canal elegido y *x* el número de grupo al que se desea enviarlo. Con el método [vis(se pueden abrir y visualizar las instancias y con el *bang* del *patch* madre se "crean los grupos", es decir se crea el objeto del *slider* maestro y su *trigger*, lo cual es sumamente necesario para mantener un orden de creación coherente con los mensajes *connect.*

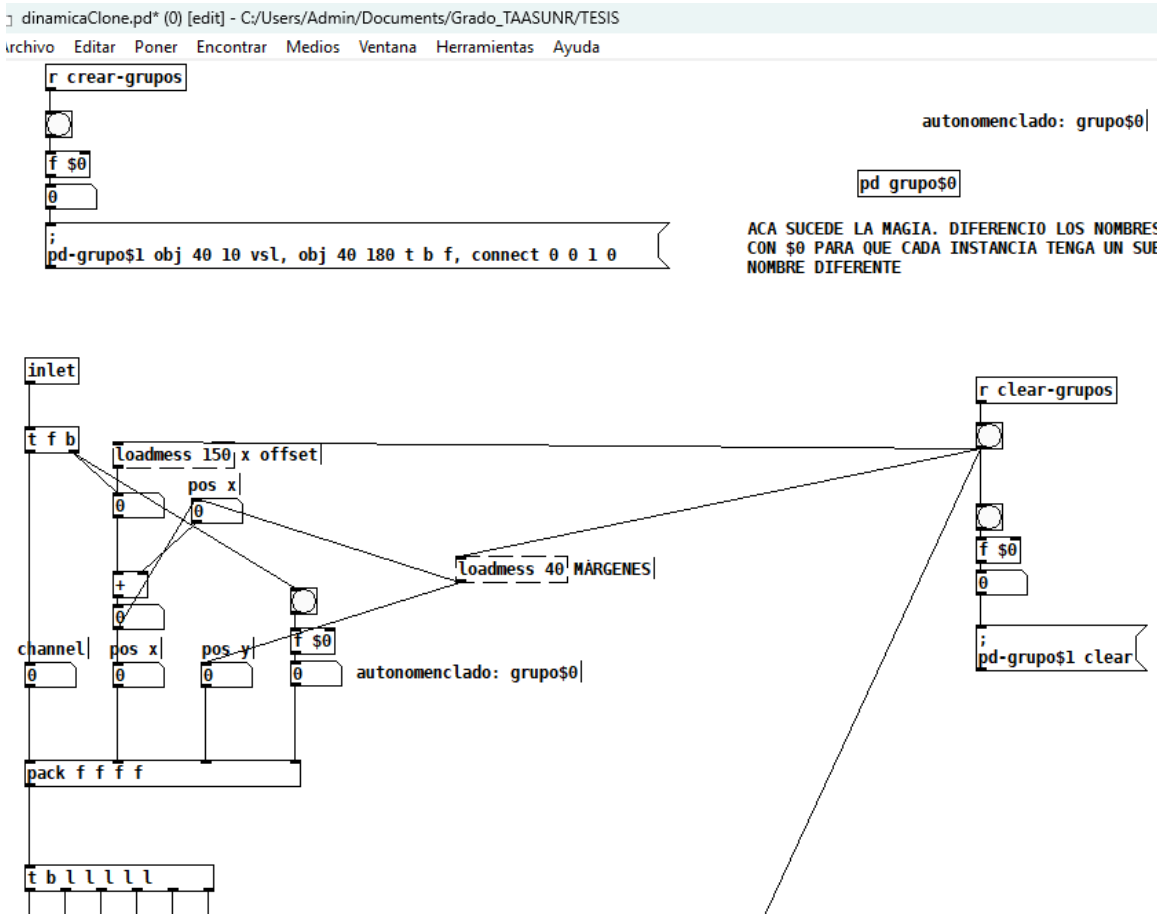

Figura 36. Patch de *Pure Data: PrototipoDeExtension_002*. Abstracción clonada *dinamicaClone*, instancia 0. Foto 1.

Varias pruebas fueron necesarias para que este prototipo sea funcional, debido a la complejidad de estar trabajando a la par con programación dinámica y clonación. En última instancia se decidió usar el valor de ID particular de cada canvas, representado por $0 para nomenclar a cada grupo, los cuales están en un *subpatch* dentro de cada instancia clonada. Se pensaba inicialmente que un mensaje del estilo [pd-grupo$0 clear(podría ser suficiente para que se interpretara esa variable dinámica pero eso no fue viable. Se necesitó ingresar el valor de canvas como una variable desde el *inlet* del mensaje.

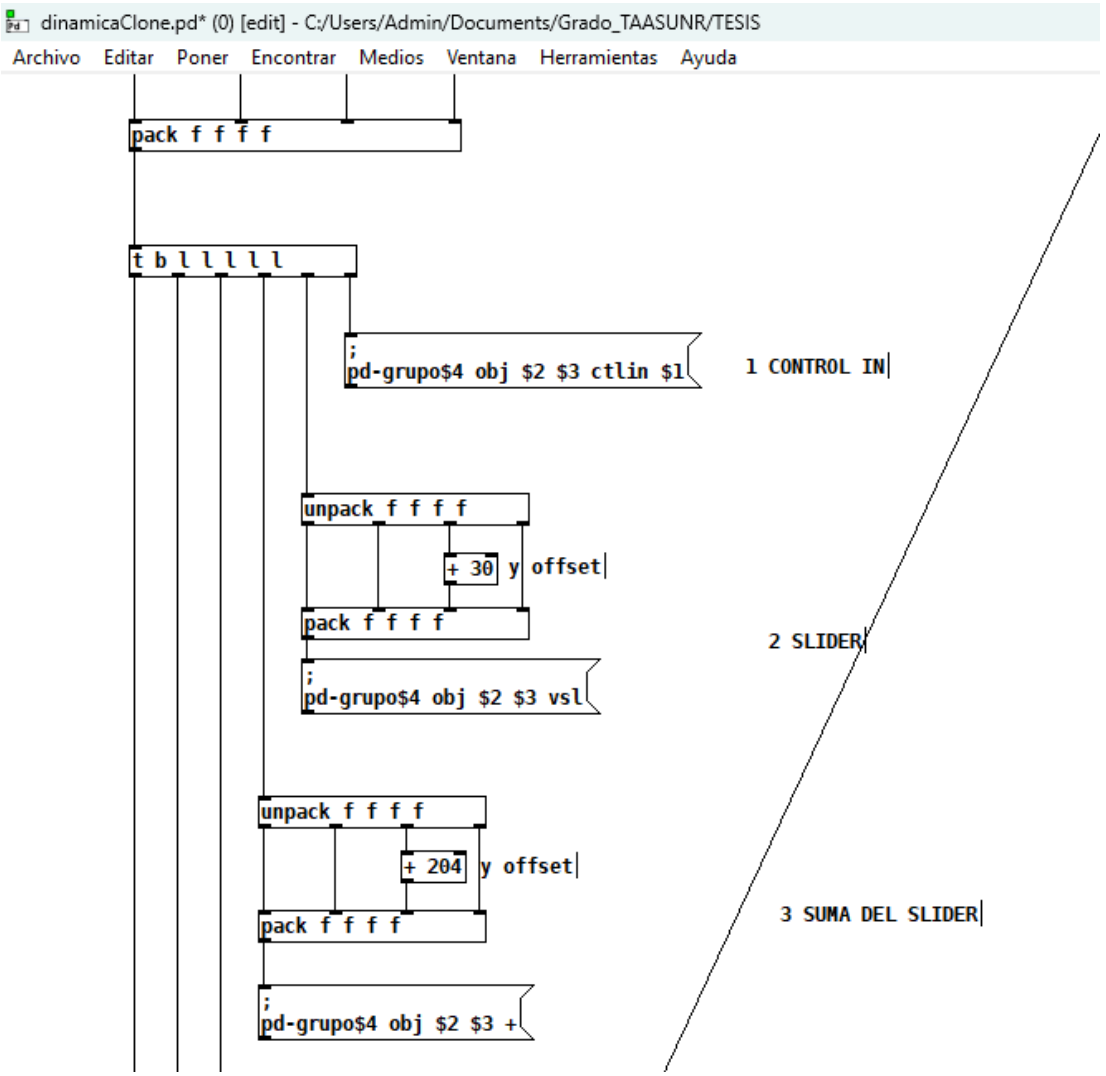

Figura 37. Patch de *Pure Data: PrototipoDeExtension_002*. Abstracción clonada *dinamicaClone*, instancia 0. Foto 2.

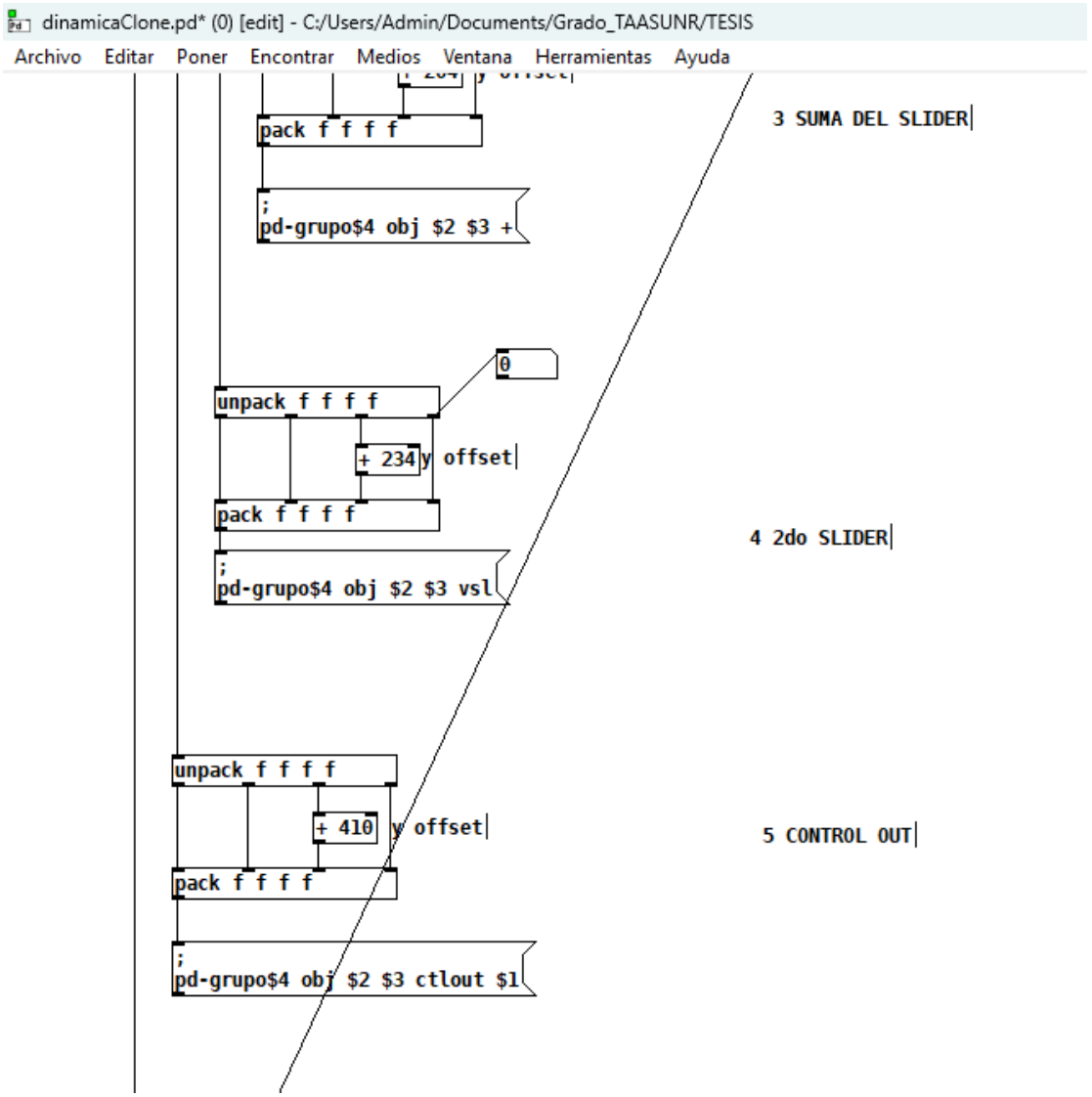

Figura 38. Patch de *Pure Data: PrototipoDeExtension_002*. Abstracción clonada *dinamicaClone*, instancia 0. Foto 3.

El orden de los objetos creados es el mismo, sólo cambia que ahora el nombre del subpatch específico debe ingresar como una nueva variable $4. Esto se vio necesario cuando sucedió que diferentes instancias estaban pisandose unas a las otras y no había diferencia entre los subpatches porque se les estaba hablando a todos o no se le estaba hablando a ninguno.

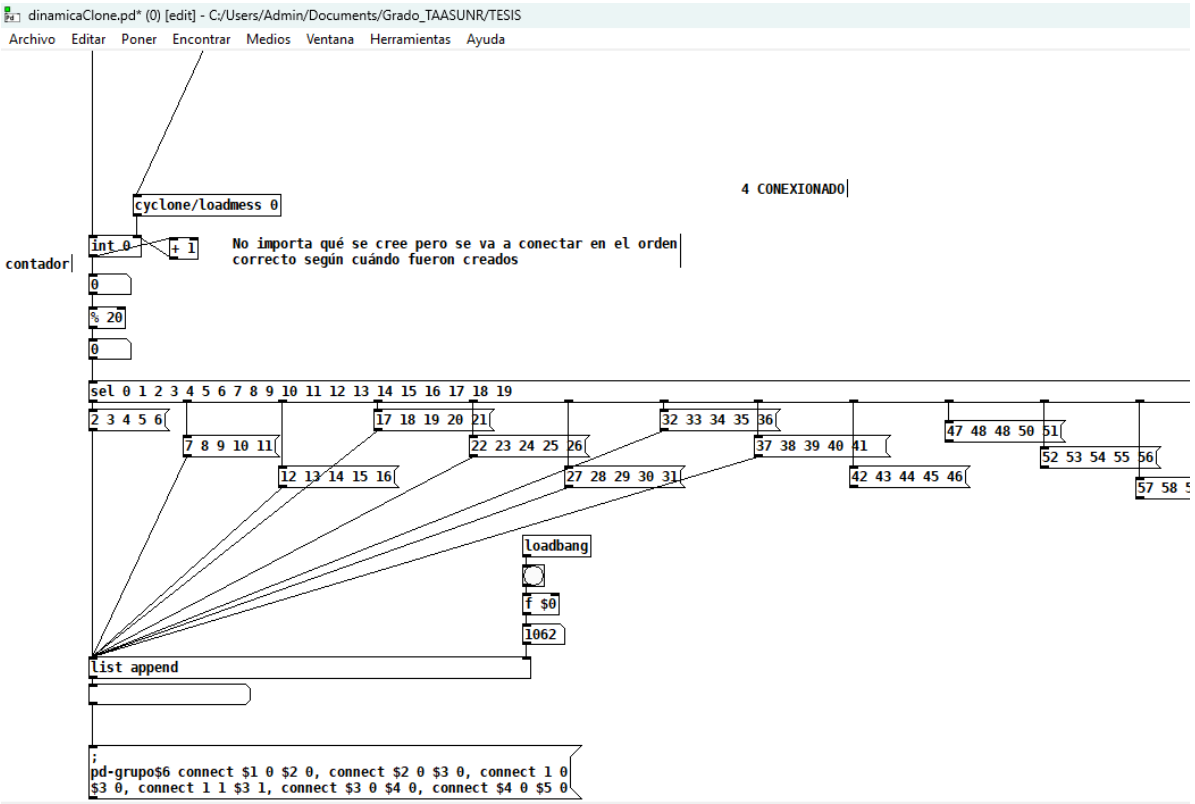

Figura 39. Patch de *Pure Data: PrototipoDeExtension_002*. Abstracción clonada *dinamicaClone*, instancia 0. Foto 4

La cuarta y última parte del *patch* es muy similar a la de los prototipos anteriores, las únicas diferencias son que se sumaron más mensajes al [select] para permitir grupos de mayor tamaño y que se necesitó un objeto [list append] para poder adosar como sexta variable ($6) el número de ID particular del subpatch.

Tercer capítulo
Presentación de los resultados finales y conclusiones

Tercer capítulo. Presentación de los resultados finales y conclusiones

Prototipo final y resultados

Gracias a la nueva búsqueda realizada luego de ser fallidos los primeros prototipos en cuanto a la comunicación se llegó a la conclusión de que la forma empleada por la consola utilizada para realizar las pruebas en el EMEA, la SQ6 de la marca Allen & Heath, de enviar la posición de los faders como mensaje MIDI es por medio de mensajes del tipo NRPN. Como se dijo anteriormente en el marco teórico este tipo de subprotocolo compone un mensaje de cuatro bytes que trabajan en conjunto: los dos primeros definen el número de parámetro a través del CC 99 y el 98 y los dos últimos definen el valor transmitido a través del CC 6 y el 38. Por los CC 99 y 6 recibimos el byte más significativo (*most significant byte*), donde el peso de cada bit es de 128 y por los CC 98 y 38 encontramos el byte menos significativo (*less significant byte*) donde cada bit pesa 1. De este modo podemos contar con una profundidad de 14 bits, a diferencia de los 7 de los mensajes MIDI sencillos de un sólo byte, y tenemos por lo tanto de valores del 0 al 16.383. Estos valores numéricos de *Control Change* definidos para el NRPN coincidían exactamente con los mensajes recuperados en la consola de *Pure Data.*

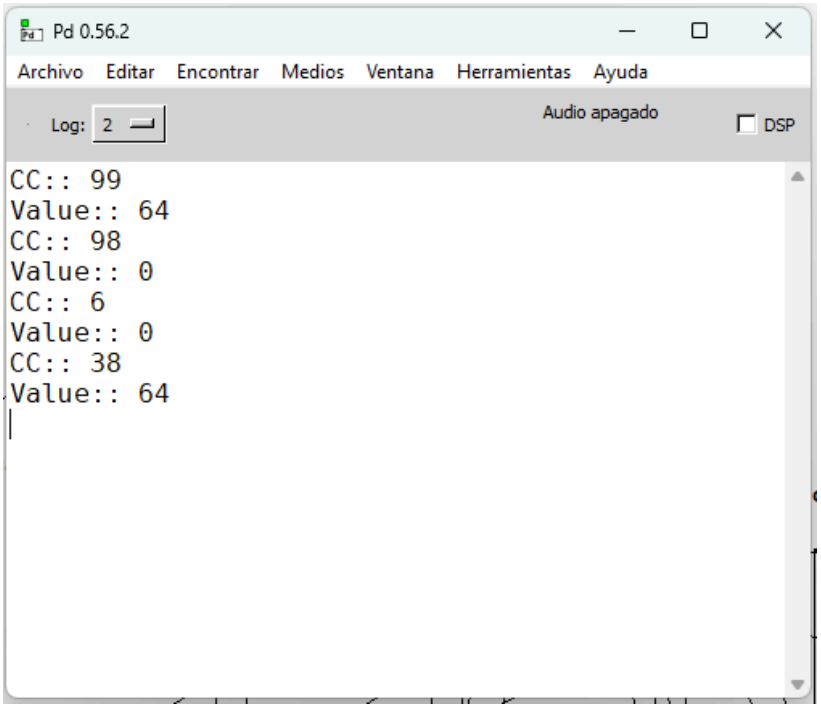

Figura 40. Confirmación de recepción de mensajes NRPN. Impresión en consola

Para poder interpretar los mensajes NRPN se decidió utilizar un software externo que hiciese de intermediario entre *Pure Data* y la consola de audio: MIDI-OX, un programa de Windows creado por Jamie O'Connell y Jerry Jorgenrudis. Funciona como una herramienta de diagnóstico y como una librería de *System Exclusive*, puede filtrar y *mapear* mensajes MIDI crudos. Muestra mensajes entrantes y los envía a donde se le indique, se pueden generar mensajes MIDI usando el teclado de la computadora o el panel de control. MIDI-OX es un software de tipo *freeware* y se encuentra protegido por derechos de autor.

Como el ingreso de información no fue tan sencillo como se esperaba se tuvo que modificar la plantilla base original (*PrototipoDeExtension_000*) para poder interpretar los mensajes de entrada NRPN y reinterpretar los mensajes previo a su salida. Para esto fue de suma utilidad el objeto [route] que permitió filtrar los valores que ingresaban por los diferentes controladores.

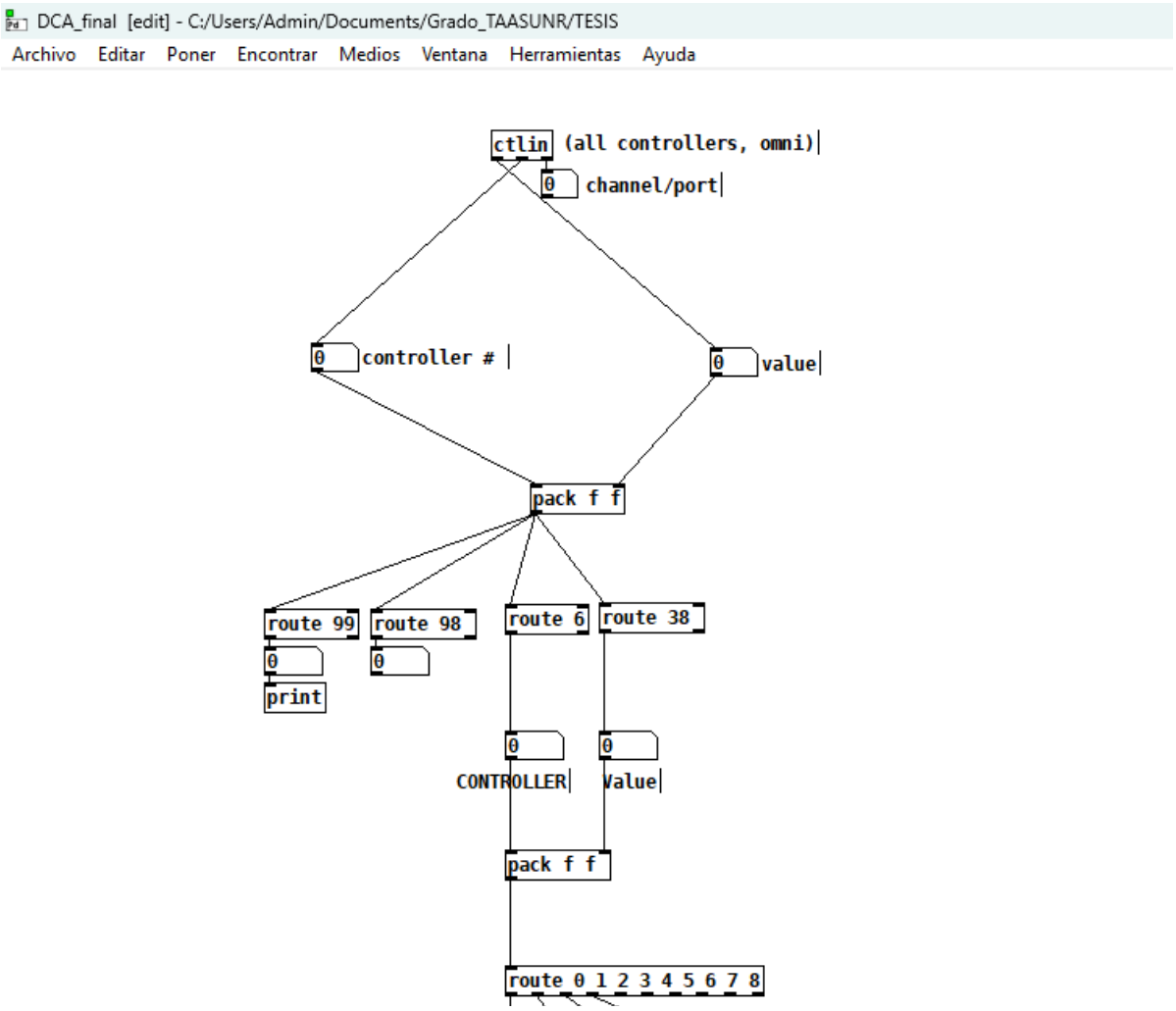

Figura 41. Patch de *Pure Data: PrototipoDeExtension_final*. Subpatch *DCA_final*. Foto 1

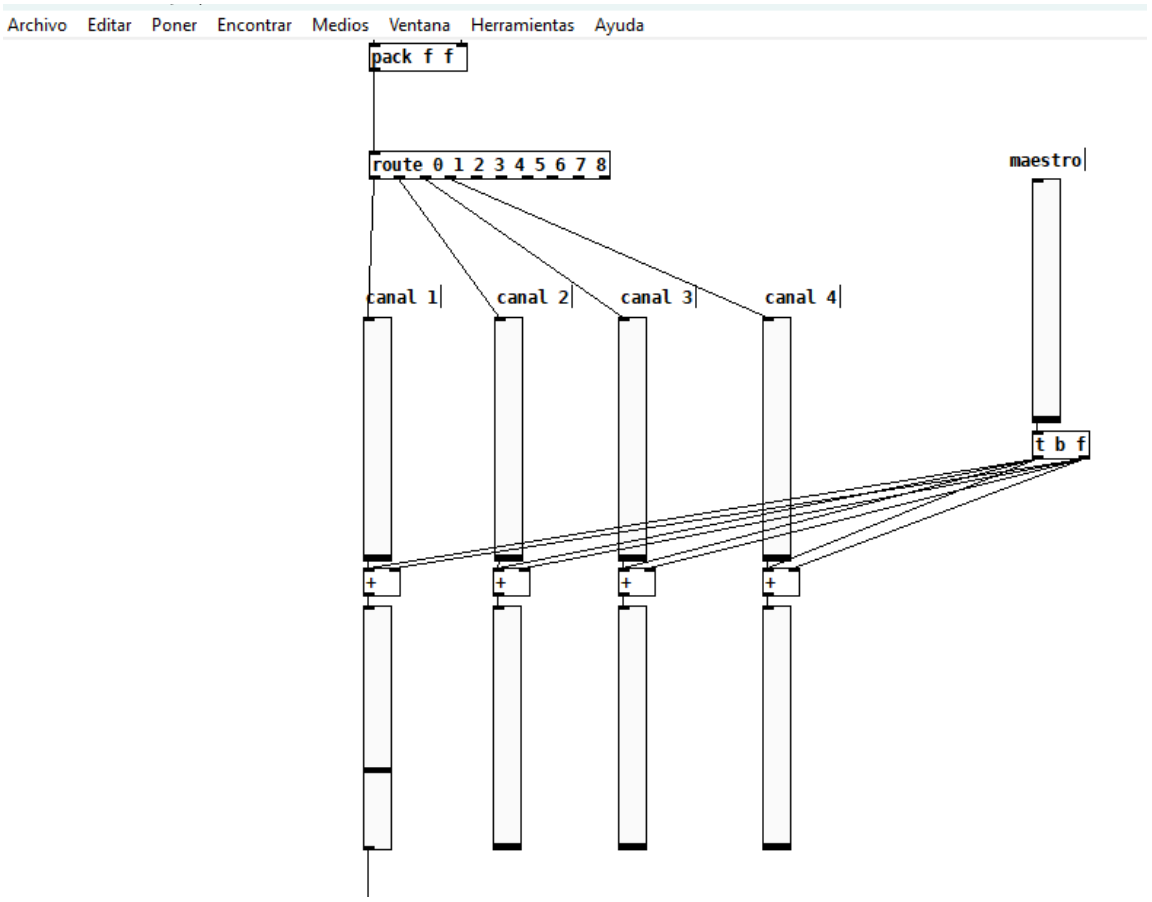

Figura 42. Patch de *Pure Data: PrototipoDeExtension_final*. Subpatch *DCA_final*. Foto 2

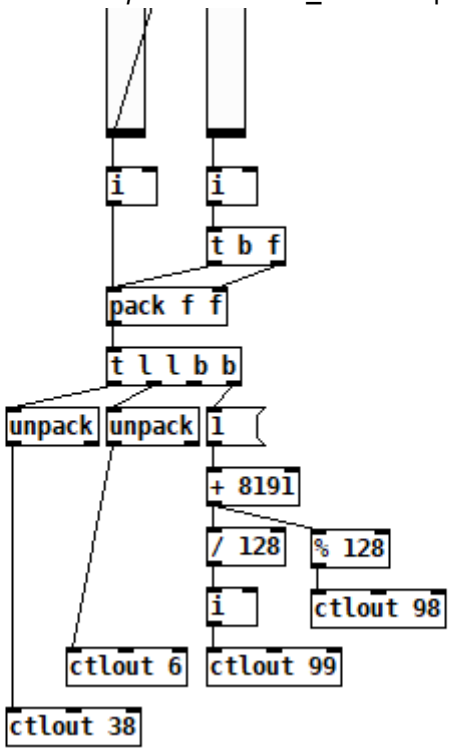

Figura 43. Patch de *Pure Data: PrototipoDeExtension_final*. Subpatch *DCA_final*. Foto 3

A la salida se necesitó interponer el software mencionado MIDI-OX para poder recodificar el mensaje MIDI NRPN para enviar hacia la consola. No se había

previsto en las primeras instancias de proyecto de investigación la necesidad de usar un elemento externo en la cadena, sin embargo no dificulta la aplicación y replicabilidad ya que se encontró una opción que sea también de software libre y liviana. De este modo se concluyó la plantilla final, ahora disponible para aplicar en el *subpatch* de programación dinámica y luego dentro de una instancia de clonación.

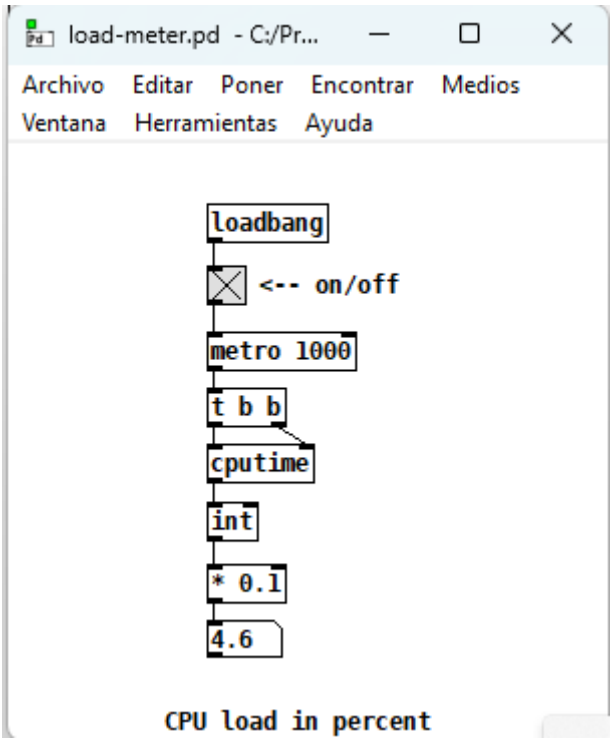

Figura 44. Carga del CPU

Con esta herramienta brindada por PD se pudo ir constatando a lo largo del proceso de prototipado y de prueba que la carga del CPU de la cual era responsable el entorno y el procesamiento el DSP no superara un 10% de la capacidad total. Como el procesamiento de señal no estuvo presente en el proyecto no hubo problemas de sobrecarga, el procesamiento de datos y cómputos no conlleva un problema de cortes en la comunicación o latencia pertinentes.

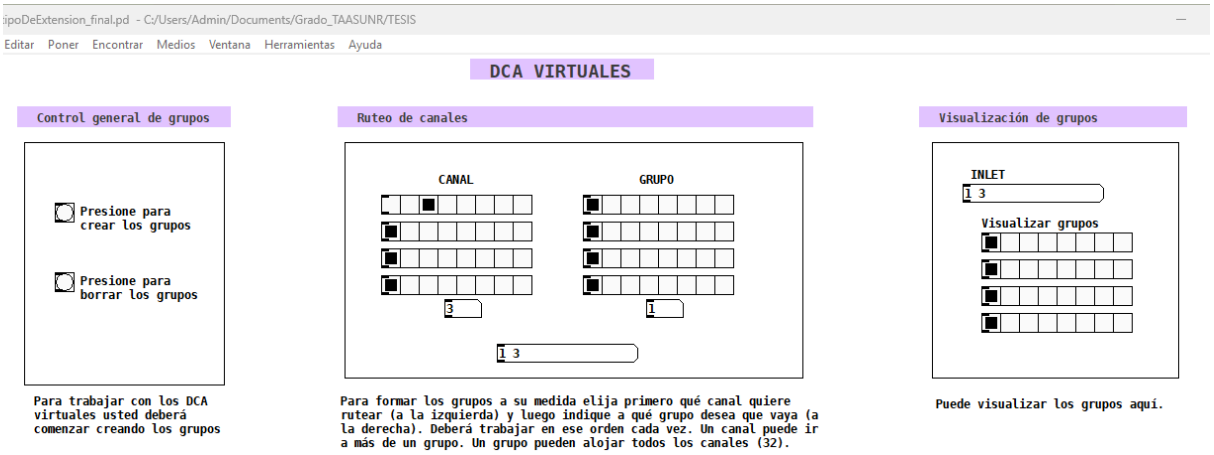

Figura 45. Patch de *Pure Data: PrototipoDeExtension_final*. *Patch* madre.

En la versión final se pueden visualizar tres *subpatches* principales: control general de grupos, ruteo de canales y visualización de grupos. En el primero se encuentran los dos botones o *bangs* que permiten crear los grupos, es decir crear y conectar los primeros objetos que se necesitan independientemente de qué canales ingresen, y borrar todo lo que se encuentre en los grupos. No se encontró forma de borrar discriminadamente, por lo que el segundo *bang* vacía todos los grupos y por completo.

En el segundo *subpatch* se habilita la posibilidad de decidir cómo se quieren crear los grupos: hay 32 grupos disponibles con 32 canales disponibles cada uno (medida que supera en gran medida a todos los DCA presentes en consolas en el mercado actual, donde suele haber 8 grupos disponibles). Para poder seleccionar qué canal quiero en cada grupo se debe seguir la secuencia de primero hacer *click* en un canal, dentro de los selectores que a su vez visualizan el número de canal seleccionado en la caja de números ubicada debajo, y luego *clickear* en el grupo al que se quiere enviar ese canal. Si por error se selecciona un canal no deseado se puede volver atrás porque el evento que detona el envío y la creación del código es el grupo, pero en el caso de equivocarse en este segundo paso se deberá apretar el botón de *clear* y volver a empezar.

A través del tercer subpatch, denominado “visualización de grupos”, se puede acceder a cada instancia de clone con un selector para poder observar cómo se componen las voces de clonación y dentro de cada voz visualizar el subpatch llamado *grupo$0* que es donde se aloja el grupo de esa instancia en particular.

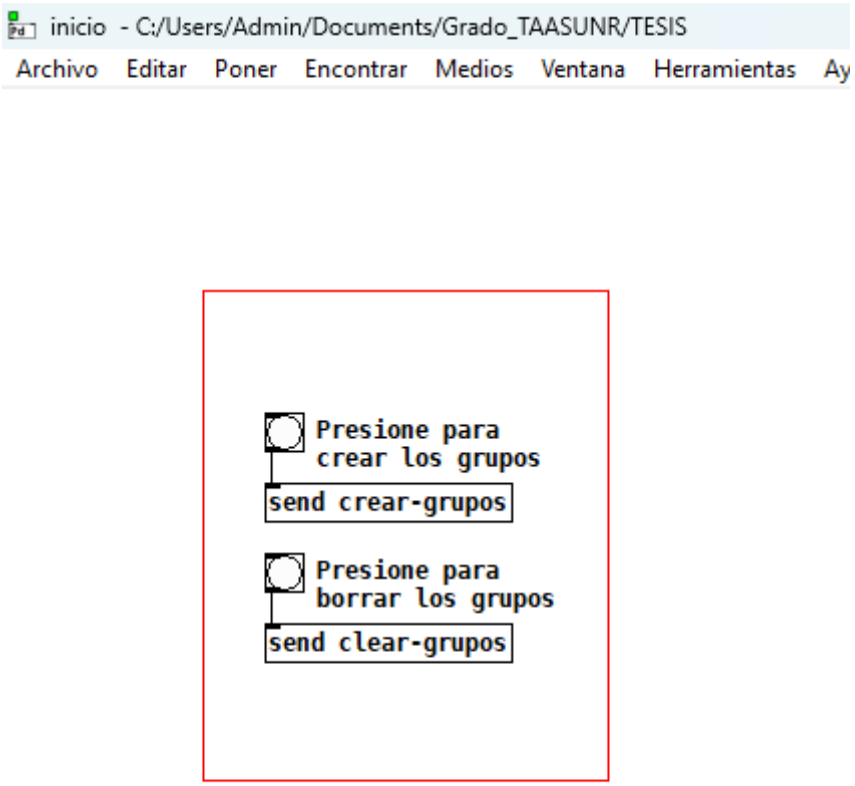

Figura 46. Patch de *Pure Data: PrototipoDeExtension_final*. *Subpatch* inicio (bajo el título “Control general de grupos” en el *patch* principal)

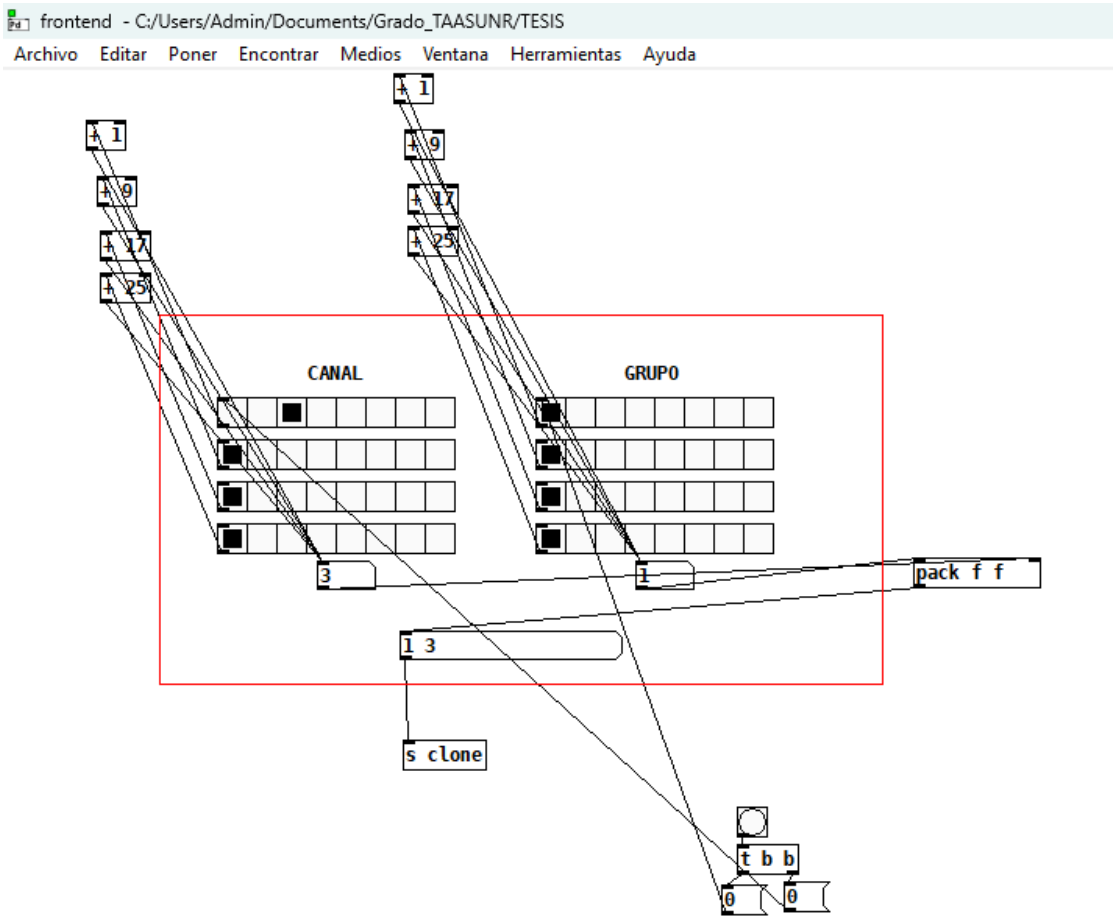

Figura 47. Patch de *Pure Data: PrototipoDeExtension_final*. *Subpatch* frontend (bajo el título “Control general de grupos” en el *patch* principal)

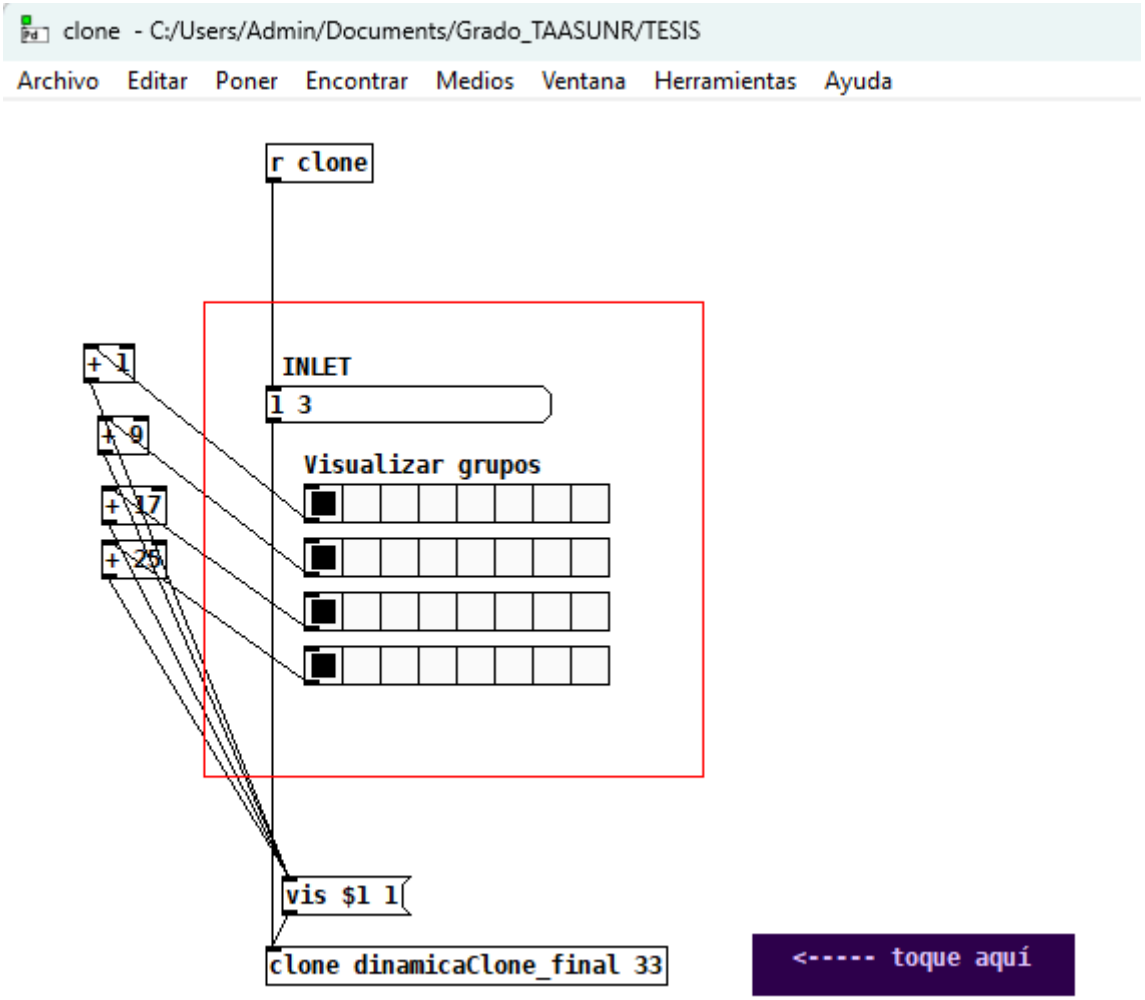

Figura 48. Patch de *Pure Data: PrototipoDeExtension_final*. *Subpatch* clone (bajo el título "Visualización de grupos" en el *patch* principal)

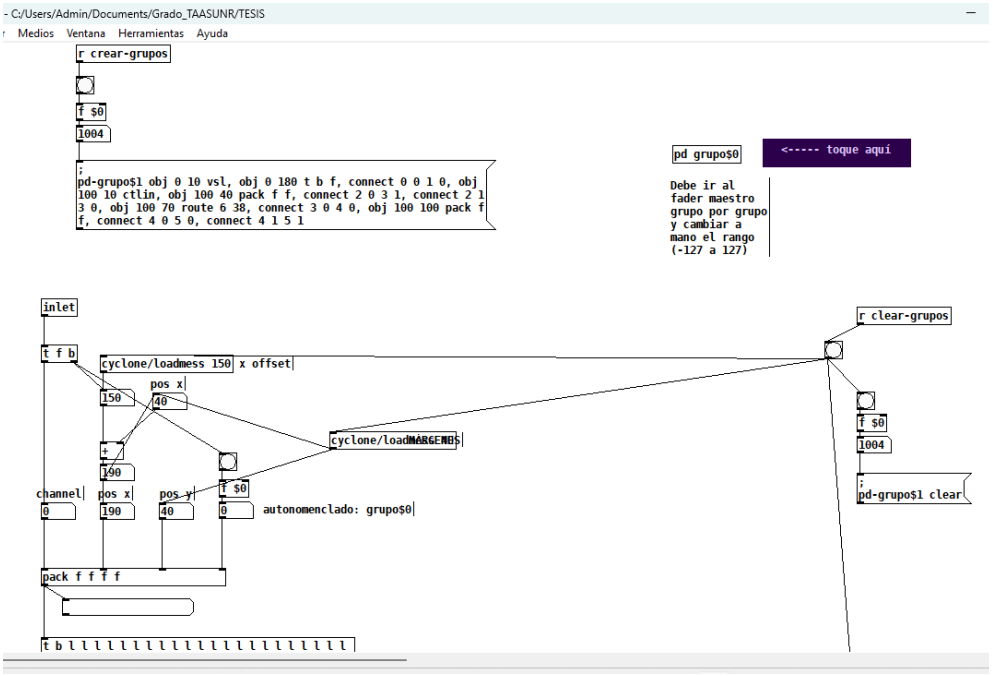

Figura 49. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 1

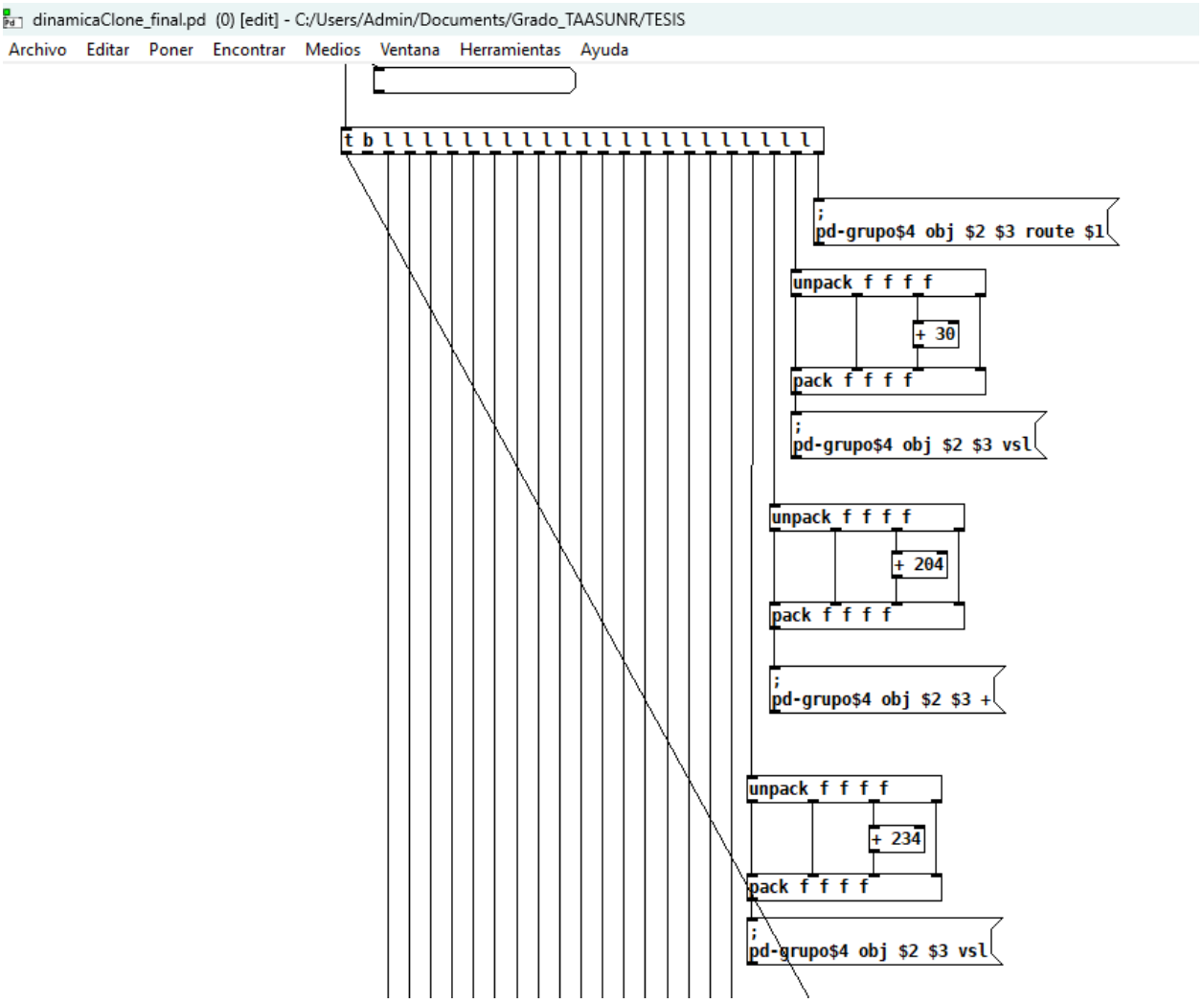

Figura 50. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 2

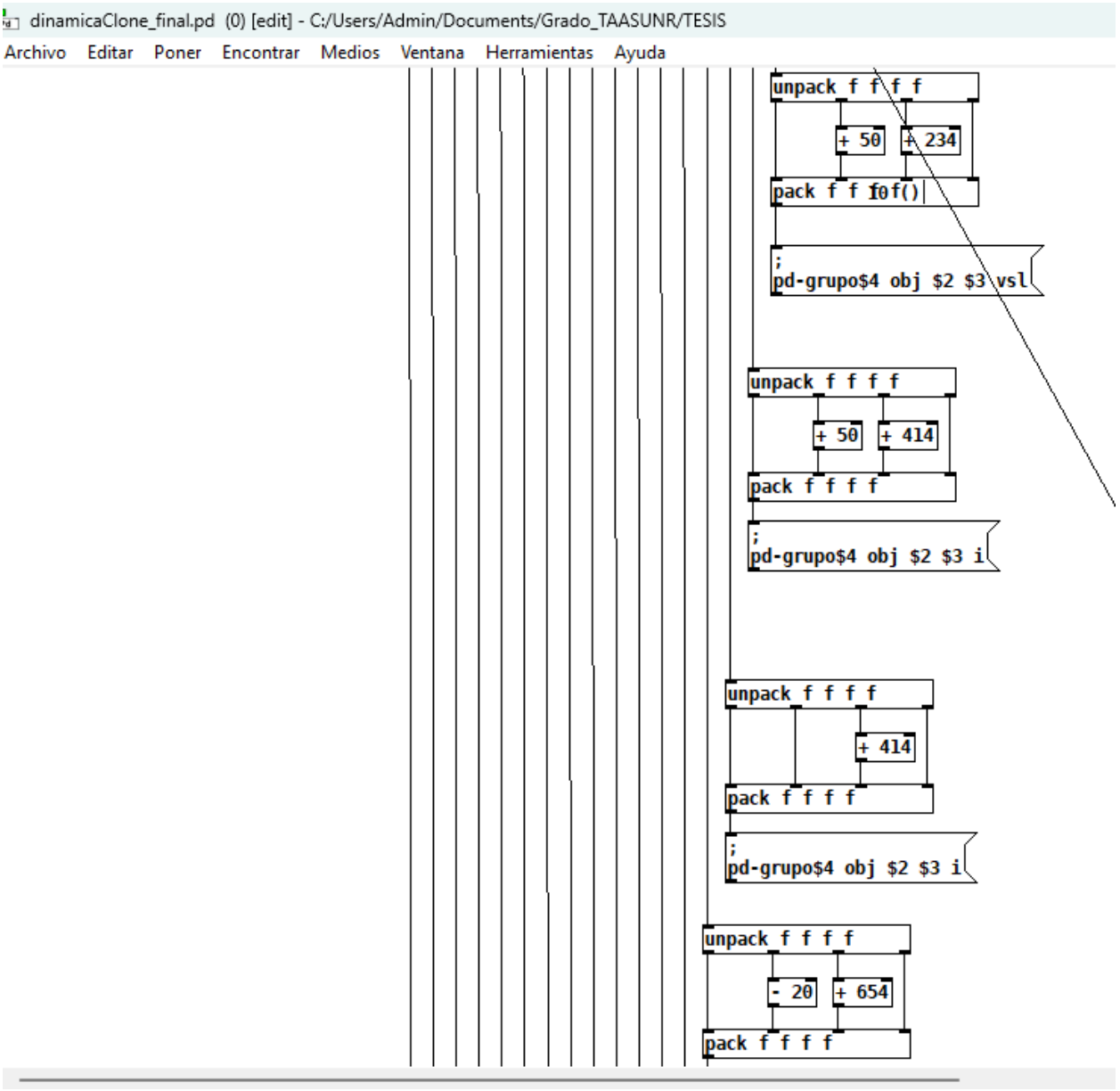

Figura 51. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 3

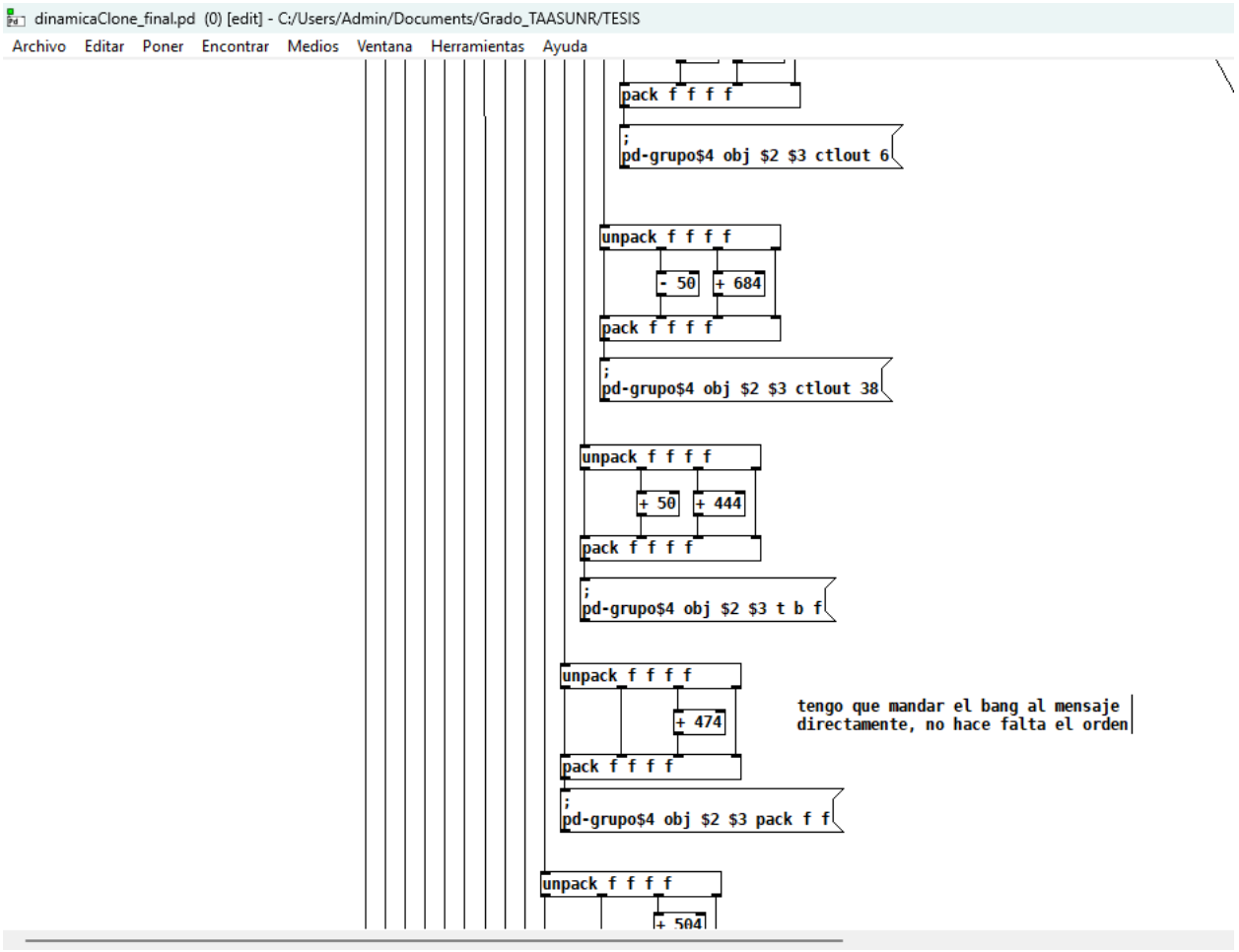

Figura 52. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 4

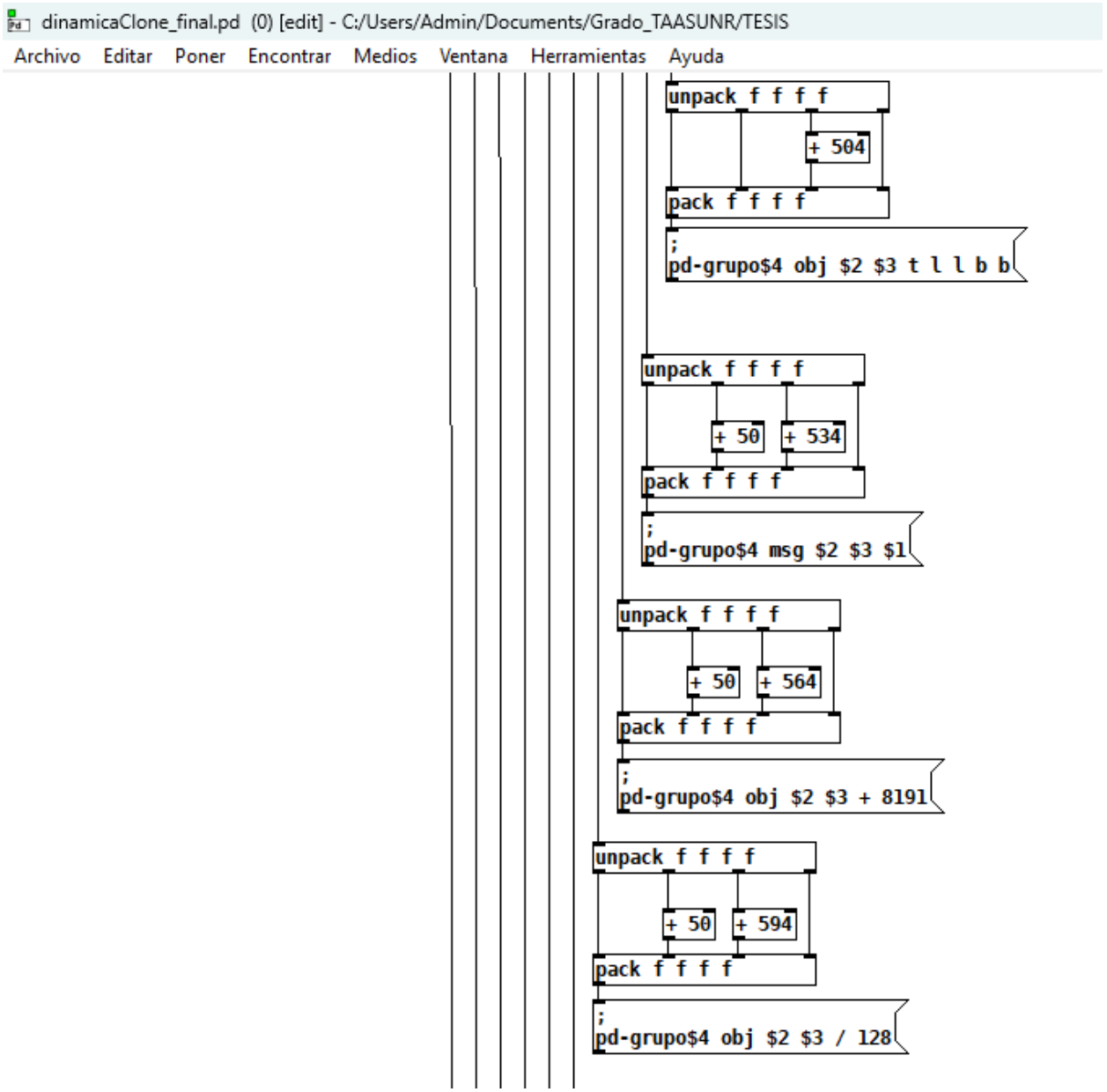

Figura 53. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 5

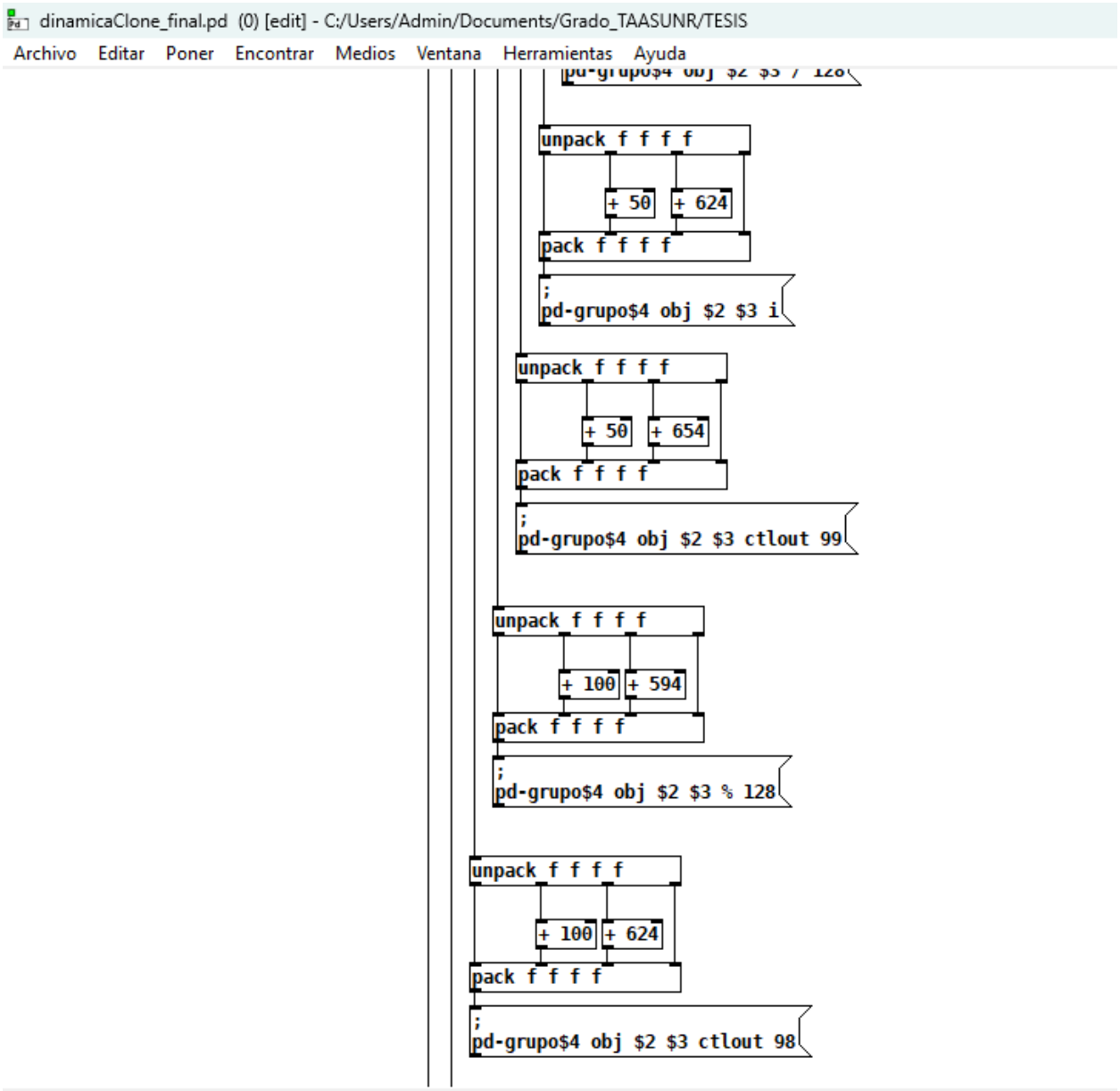

Figura 54. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 6

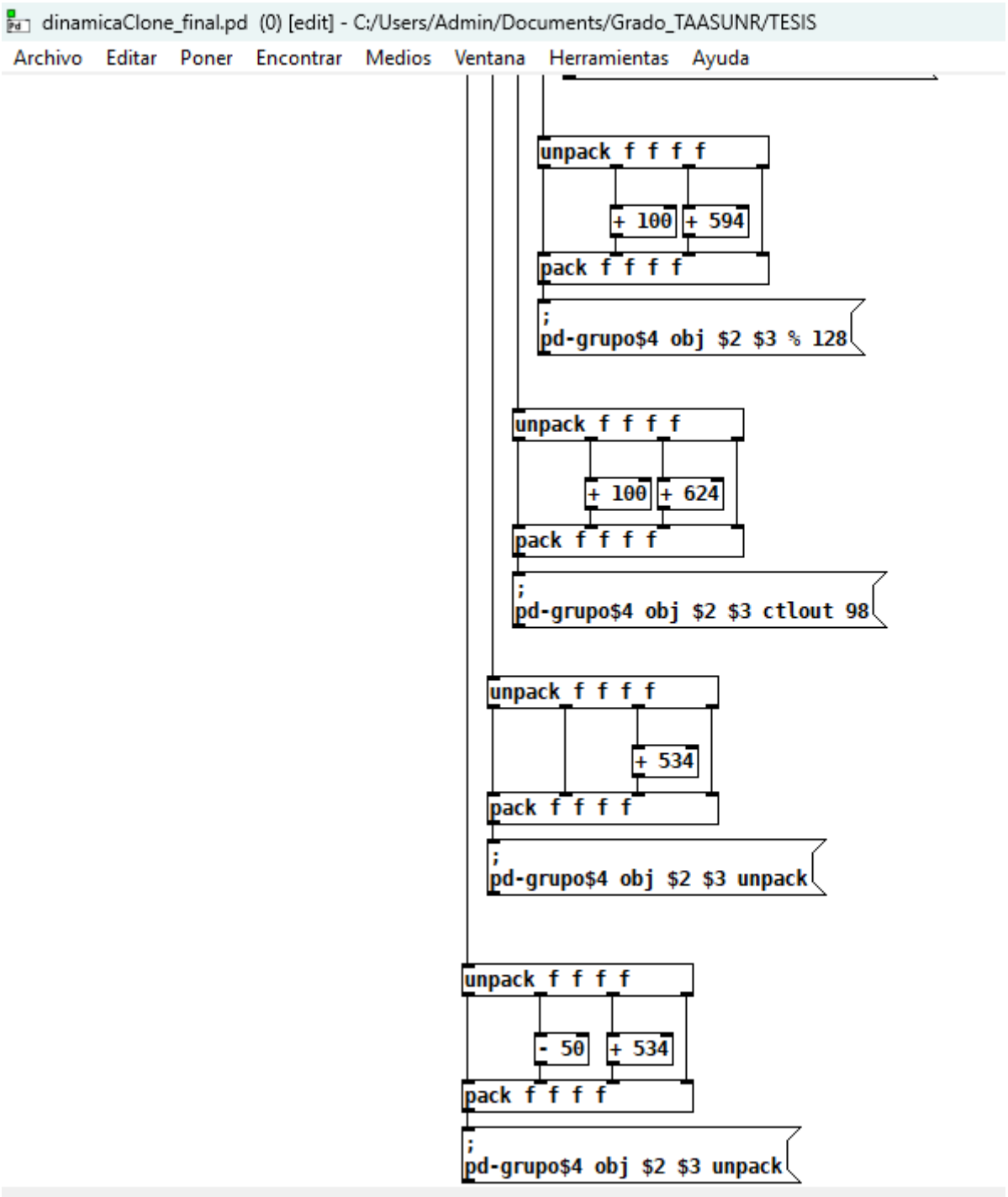

Figura 55. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 7

Establecida una comparación con la abstracción anterior (dinamicaClone) se puede observar que la lógica de programación se mantiene, el uso de un *trigger* secuencial con listas y un *bang* final sigue presente pero se ve extendido de seis (6) *outlets* a veintidós (22), y la etapa del conexionado (vista a continuación) se

complejizó.

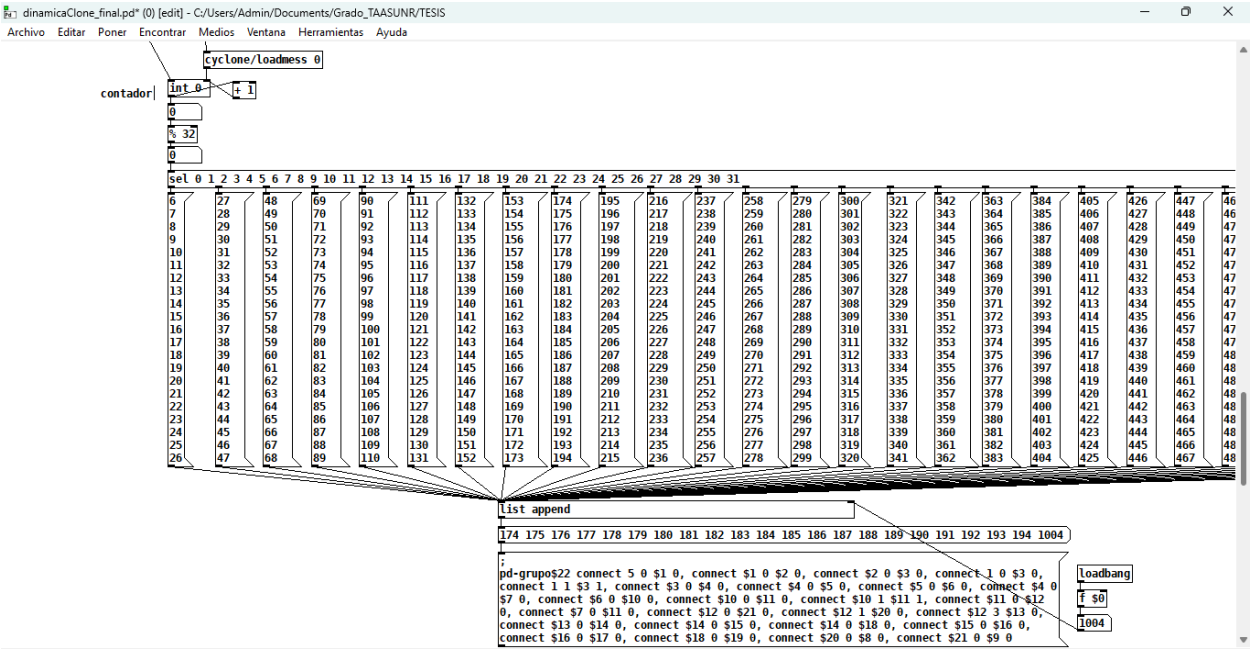

Figura 56. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 8

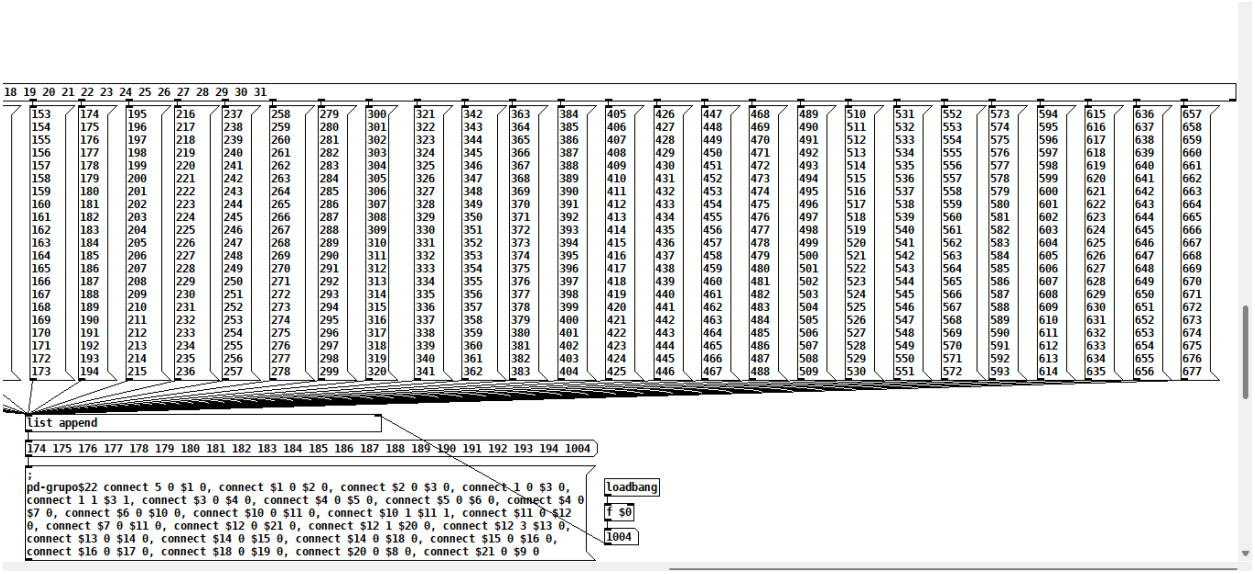

Figura 57. Patch de *Pure Data: PrototipoDeExtension_final*. Primera instancia de clonación de la abstracción dinamicaClone_final. Foto 9

En la etapa de´la abstracción donde el método de programación dinámico realiza el conexionado entre los objetos creados necesitó de un contador con treinta y dos (32) valores, de 0 a 31, una etapa de filtrado y luego treinta y dos (32)

mensajes (por la cantidad de canales máxima total dentro de cada grupo) con veintiún (21) valores ordenados, arrancando por el 6 y terminando en el 677. En el caso de que un grupo cuente con la cantidad máxima posible de canales el índice del primer objeto que se debe conectar en esta instancia es el 6, ya que los anteriores se crean y conectan primero y de manera independiente con el botón de *crear,* y el último objeto del último canal tendría el índice 677. Esos valores son almacenados en la lista, en donde se agrega al final el valor de $0 del canvas como el valor número 22, que luego alcanza el mensaje dinámico donde cada valor ingresado es interpretado como una variable.

A modo ilustrativo de cómo quedan finalmente los grupos se eligió que el grupo 3 tenga alojados los canales 1, 3, 5 y 7. (Mostrado a continuación).

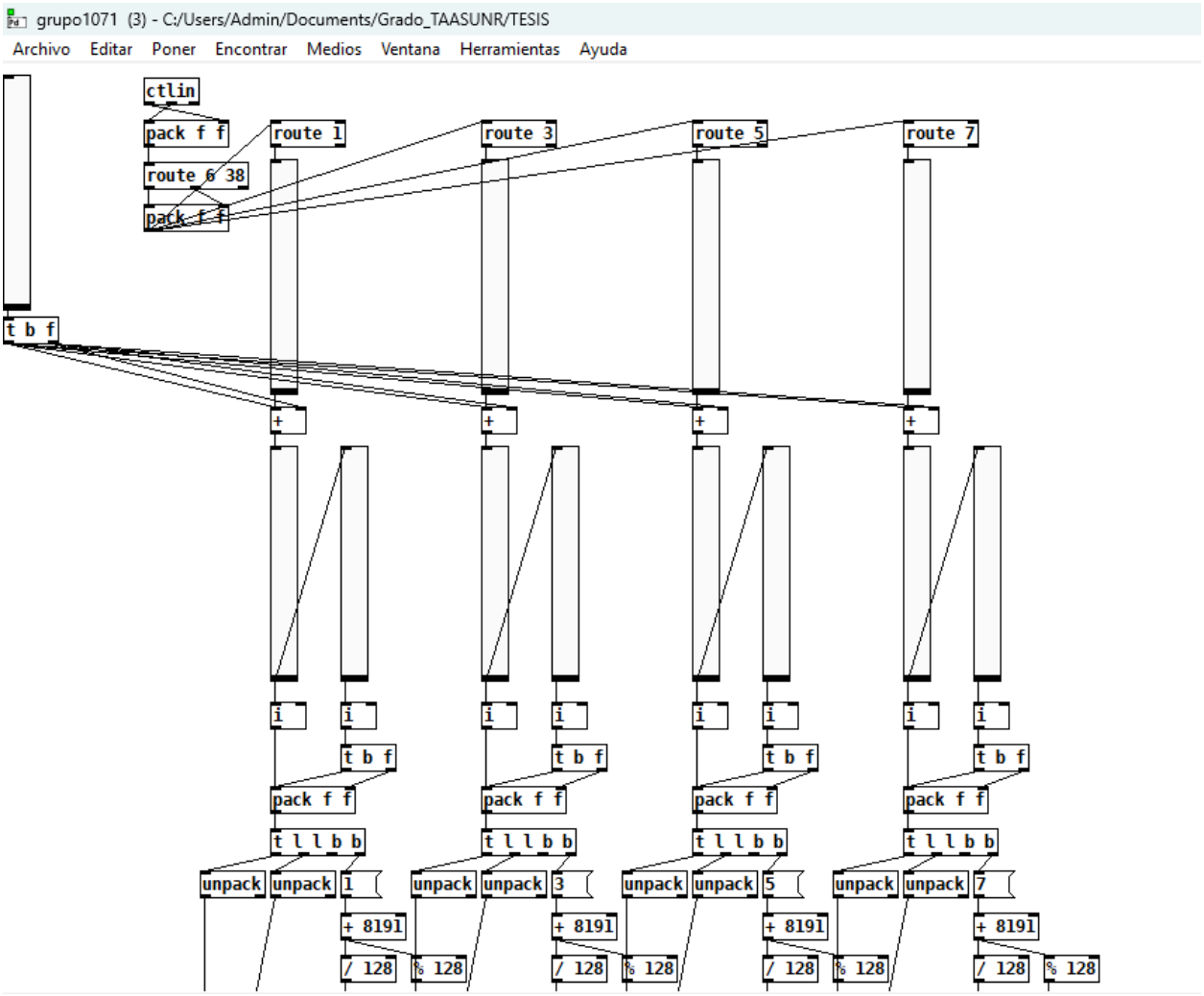

Figura 58. Patch de *Pure Data: PrototipoDeExtension_final*. *Subpatch* grupo$0, dentro de la abstracción dinamicaClone_final. Foto 1

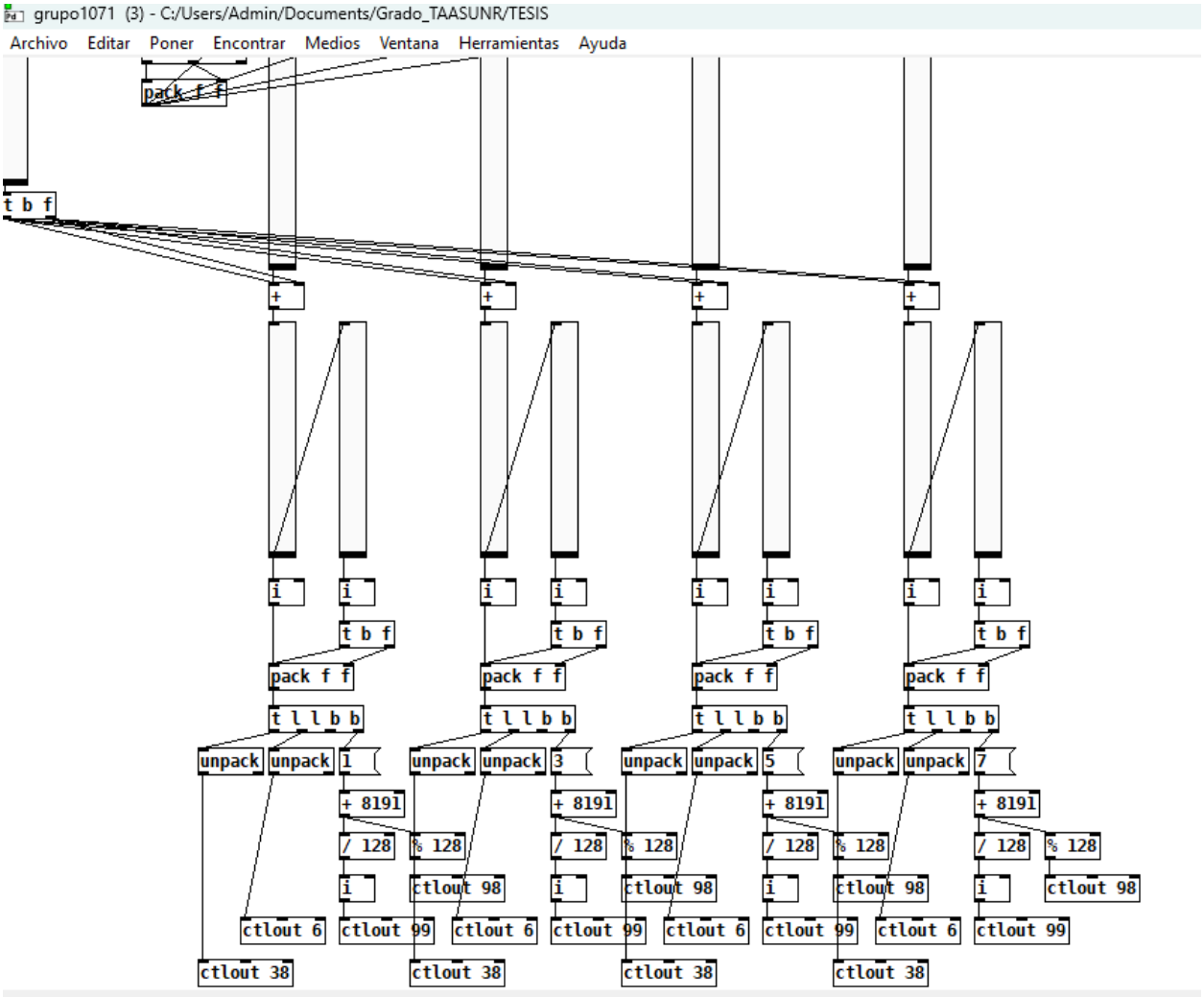

Figura 59. Patch de *Pure Data: PrototipoDeExtension_final*. *Subpatch* grupo$0, dentro de la abstracción dinamicaClone_final. Foto 2

La puesta en práctica

El hardware utilizado para realizar las pruebas estuvo constituído de una laptop Dell Inspiron al principio y una HP ProBook después, ambas con Windows 11, junto con la consola de Allen & Heath SQ6. La conexión entre computadora y mesa de mezcla se realizó con un cable USB A/b usando la conexión formato A en la pc y el b para la consola. Las versiones de los softwares utilizadas fueron las últimas publicadas en ambos casos: para *Pure Data* la versión PD 0.56.2 y la 7.0.2 de MIDI-OX.

Conclusiones: reflexión crítica del proceso y de los aportes de este trabajo

El presente trabajo tuvo como propósito principal el desarrollo de una herramienta capaz de ampliar las prestaciones de las consolas de mezcla digitales mediante un *patch* desarrollado en el entorno de programación gráfica Pure Data, integrando capacidades de control externo vía MIDI y procesamiento interno en tiempo real. Este proyecto se planteó como una investigación de desarrollo aplicado, en la que la validación técnica y la exploración del comportamiento real de los sistemas ocupan un rol central: a lo largo del proceso, el eje principal estuvo puesto en establecer una comunicación fiable entre la consola y la computadora, dado que de esta dependía la posible aplicación de todas las funcionalidades desarrolladas.

En términos generales, los objetivos propuestos fueron ampliamente alcanzados. Se logró implementar un conjunto de DCA (Digital Controlled Amplifier) virtuales capaces de agrupar y controlar el nivel de hasta treinta y dos (32) *faders* manteniendo las relaciones relativas originalmente tomadas de las posiciones enviadas desde la consola. Se consiguió que los valores actualizados de los *faders*, una vez realizado el procesamiento del grupo con el *fader* maestro, sean enviadas desde Pure Data, con el *software* MIDI-OX como intermediario para poder codificar el mensaje MIDI NRPN de cuatro eventos CC (Control Change) empaquetados, a la consola para ser recuperados por los *faders* motorizados. Se consiguió, además, operar con una latencia imperceptible para el usuario gracias al bajo consumo del entorno y a la eficiencia del código desarrollado. Se otorgó particular interés a realizar los objetivos con la menor cantidad de objetos posibles a la hora de clonar instancias de la abstracción, donde se llegó a trabajar con hasta treinta y dos (32) instancias en simultáneo. Asimismo, en consecuencia del uso de la programación dinámica y del ruteo de canales se logró desarrollar una interfaz flexible y ajustable a distintos modos de trabajo. Por otro lado, el objetivo específico vinculado al establecimiento de una base normativa de comunicación entre la consola, la computadora resultó ser uno de los aportes más relevantes del proyecto. Otorga nuevo conocimiento a la comunidad académica y al ámbito de profesionales del audio facilitando la creación de nuevas herramientas e investigaciones de desarrollo relacionadas. En cuanto a las automatizaciones, si bien fueron diseñadas y parcialmente implementadas, no se alcanzó un estado de prueba y validación

suficiente que permitiera considerarlas completamente finalizadas. No obstante, su estructura funcional quedó planteada por lo que su desarrollo posterior resulta abordable.

Las dificultades encontradas durante el proceso – el uso de mensajes NRPN, desconocidos antes de comenzada la investigación– constituyeron el desafío técnico más importante del trabajo. Resolver estas problemáticas encontradas permitió sentar las bases de un esquema de comunicación sólido y replicable, habilitando la posibilidad de expandir la herramienta hacia nuevas funcionalidades o nuevos sistemas en el futuro. A través de la experiencia directa se observaron parte de los desafíos característicos de las investigaciones de desarrollo aplicado. Entre ellos se pueden mencionar la necesidad de trabajar con información técnica incompleta o inicialmente errónea, la obligación de reformular decisiones de diseño a medida que se revelan nuevas condiciones del sistema y en particular para esta investigación las dificultades propias de integrar dispositivos y entornos heterogéneos. Estas experiencias enriquecieron el resultado del proyecto y aportaron una visión más profunda sobre las dinámicas reales del desarrollo tecnológico aplicado, proporcionando herramientas metodológicas valiosas para futuras instancias de investigación y producción.

Se incorporaron aprendizajes significativos en términos de metodología de programación. A lo largo del trabajo de prototipado fue necesario adoptar prácticas sistemáticas como la utilización de un repositorio para documentar y resguardar las diferentes etapas del proyecto, posibilitando la recuperación de versiones previas y la creación de ramas de experimentación sin comprometer la estructura principal. Se buscó siempre descomponer cada problema en su unidad conceptual más simple y construir soluciones de manera modular, lo cual permitió integrar *patches* más complejos sin la acumulación de errores de etapas anteriores. Este enfoque incremental, junto con el análisis de trabajos previos —propios y ajenos—, resultó fundamental para comprender distintas estrategias de resolución, comparar lógicas internas y mejorar la toma de decisiones durante el diseño.

Por último, gracias a la experiencia intensiva de desarrollo con Pure Data se logró reconocer las limitaciones propias del entorno. Al estar orientado al prototipado más que a la compilación de aplicaciones estables, emergieron diversos

inconvenientes que afectan el funcionamiento de la herramienta en escenarios reales: inestabilidad en el desempeño, artefactos en el movimiento de los *faders* motorizados, comportamientos erráticos en el cableado virtual MIDI, necesidad de software externo para manejar paquetes NRPN ante la ausencia de objetos específicos, y un diseño de interfaz poco adecuado para el uso en vivo ya que no facilita la creación de elementos gráficos para un flujo de trabajo intuitivo y flexible. Estas dificultades no invalidan lo realizado: funcionan como consideraciones clave para la planificación del siguiente paso lógico del proyecto. En este sentido, sería esperable que en una próxima instancia se busque desarrollar la herramienta en el lenguaje de programación basado en código C++, con Visual Studio como IDE y utilizando el *framework* JUCE, con el propósito de obtener una aplicación *standalone* robusta, estable y diseñada específicamente para audio profesional en vivo. Una vez alcanzada esta implementación, se prevé una instancia de validación con consolas de distintas marcas para determinar si existen variaciones en los valores utilizados para direccionar parámetros mediante NRPN. La normalización de estas diferencias permitirá construir un sistema con una alta replicabilidad. Finalmente, se proyecta una etapa de pruebas en comunidad con operadores de sonido, cuyo aporte será fundamental para evaluar la usabilidad, el desempeño en situaciones reales de trabajo y la pertinencia del diseño de la interfaz.

En síntesis, este proyecto logró cumplir sus objetivos técnicos principales, validar la viabilidad técnica del enfoque, generar un marco conceptual y un protocolo de comunicación probado que habilita un desarrollo futuro sólido y escalable. La exploración realizada constituye una instancia funcional, aunque experimental, y por ende un primer paso hacia la construcción de herramientas que optimicen la experiencia del operador desde un abordaje integral y extiendan las capacidades de los sistemas de sonido en vivo.

Bibliografía

Allen & Heath. (2023). *SQ MIDI Protocol – Issue 4*. Disponible en: https://www.allen-heath.com/content/uploads/2023/05/SQ-MIDI-Protocol-Issue4.pdf (Relevado por última vez 20/11/2025)

Brooks, W. (2006). *Music: sound: technology*. En C. Bigsby (Ed.), *The Cambridge Companion to Modern American Culture* (pp. 332–353). Cambridge University Press.

Carrascal, J. P., y Jordà, S. (2011). *Multitouch Interface for Audio Mixing*. International Conference on New Interfaces for Musical Expression. Disponible en: https://www.nime.org/archives/ (Relevado por última vez 17/9/2024)

Katz, M. (2022). *Music and Technology*. Oxford University. Oxford, Reino Unido.

Landon, P. (1995). *Art and technological sound: the revival experience*. Concordia University. Montreal, Canadá.

López Cano, R., y San Cristóbal Opazo. (2014). *Investigación artística en música. Problemas, métodos, experiencias y modelos*. Escuela Superior de Música de Cataluña. Barcelona, España.

Melczarsky, A. (2004). *Controlador remoto MIDI*. Universidad Nacional de Mar del Plata. Mar del Plata, Argentina. Disponible en: https://rinfi.fi.mdp.edu.ar/handle/123456789/843 (Relevado por última vez 19/11/2025)

Miller, P. (s.f.). *Pure Data Manual. Updated for Pd version 0.55-0*. Disponible en: https://msp.ucsd.edu/ (Relevado por última vez 2/8/2024)

Miyara, F. (1999). *Acústica y sistemas de sonido*. Universidad Nacional de Rosario. Rosario, Argentina.

Miyara, F. (2004). *Conversores D/A y A/D*. Universidad Nacional de Rosario. Disponible en: https://www.fceia.unr.edu.ar/enica3/biblio.htm (Relevado por última vez 16/9/2024)

O'Connell, J. (26 de octubre de 2018). *MIDI-OX*. MIDI-OX. Disponible en: http://www.midiox.com/ (Relevado por última vez 01/12/2025)

Polya, G. (1965). *How to Solve It* (J. Zugazagoitia, Trad.). Editorial Trillas. (Trabajo original publicado en 1944)

Puig, S. J. (1997). *Audio digital y MIDI*. Guías Monográficas. Editorial Anaya Multimedia. Madrid.

Pure Data Info. (s.f.). Disponible en: https://puredata.info/ (Relevado por última vez 01/8/2024)

Scheirman, D. (2013). *Are Audio Education Programs Keeping Pace with New Developments in Industry?* Audio Engineering Society. Disponible en: https://aes2.org/publications/elibrary-page/?id=16865 (Relevado por última vez 15/7/2024)

Swallow, D. (2010). *Live Audio: The Art of Mixing a Show*. Focus Group. Boston, Estados Unidos.

Toffler, A. (1970). *Future Shock*. Bantam Books. Nueva York, Estados Unidos.

Yamaha Corporation of America. (s.f.). *Digital mixing console LS9. Manual de instrucciones*. Disponible en: https://es.yamaha.com/es/support/manuals/?l=es&c=&k=ls9 (Relevado por última vez 18/9/2024)

ANEXO. Recopilación y descripción de los objetos utilizados dentro de *Pure Data*

Tabla de Objetos Utilizados

Objeto	***Categoría***	***Origen***	***Descripción breve***
%	*Operador lógico*	*Pure Data*	*Operador módulo: devuelve el resto de una división.*
+	*Operador aritmético*	*Pure Data*	*Suma numérica. Acepta argumentos iniciales.*
bendin	*MIDI*	*Pure Data*	*Recibe mensajes MIDI Pitch Bend.*
bendout	*MIDI*	*Pure Data*	*Envía mensajes MIDI Pitch Bend.*
bng	*GUI*	*Pure Data*	*Botón bang: emite un bang al pulsarse.*
clone	*Estructura*	*Pure Data*	*Crea múltiples instancias de una abstracción.*
ctlin	*MIDI*	*Pure Data*	*Recibe mensajes MIDI Control Change.*
ctlout	*MIDI*	*Pure Data*	*Envía mensajes MIDI Control Change.*
cyclone/loadmess	*Utilidad*	*Cyclone*	*Envía un mensaje al cargar el patch.*
declare	*Gestión*	*Pure Data*	*Declara rutas de búsqueda o librerías.*
hradio	*GUI*	*Pure Data*	*Radio horizontal: selección de opciones.*

int	*Conversión*	*Pure Data*	*Convierte flotantes a enteros.*
midiin	*MIDI*	*Pure Data*	*Recibe datos MIDI crudos.*
midiout	*MIDI*	*Pure Data*	*Envía datos MIDI crudos.*
notein	*MIDI*	*Pure Data*	*Recibe mensajes Note On/Off.*
noteout	*MIDI*	*Pure Data*	*Envía mensajes Note On/Off.*
pack	*Control*	*Pure Data*	*Empaqueta varios valores en un solo mensaje.*
pgmin	*MIDI*	*Pure Data*	*Recibe Program Change.*
pgmout	*MIDI*	*Pure Data*	*Envía Program Change.*
print	*Depuración*	*Pure Data*	*Imprime en consola.*
r	*Control*	*Pure Data*	*Recibe información enviada por un send.*
s	*Control*	*Pure Data*	*Envía información a un receive.*
sel	*Control*	*Pure Data*	*Selecciona valores exactos.*
spigot	*Control*	*Pure Data*	*Puerta lógica que deja pasar o bloquea mensajes.*
sysexin	*MIDI*	*Pure Data*	*Recibe mensajes SysEx.*
t	*Control*	*Pure Data*	*Trigger: ordena y fuerza el tipo de salida.*

tgl	*GUI*	*Pure Data*	*Toggle: produce 0/1.*
touchin	*MIDI*	*Pure Data*	*Recibe Channel Aftertouch.*
touchout	*MIDI*	*Pure Data*	*Envía Channel Aftertouch.*
unpack	*Control*	*Pure Data*	*Separa un mensaje en partes.*
vradio	*GUI*	*Pure Data*	*Radio vertical.*
vsl	*GUI*	*Pure Data*	*Slider vertical.*