

CLAM, un Entorn de Construcció Visual d'Aplicacions d'Audio

Pau Arumí

Music Technology Group
Univ. Pompeu Fabra
parumi@iua.upf.edu

David García

Music Technology Group
Univ. Pompeu Fabra
dgarcia@iua.upf.edu

Xavier Amatriain

CREATE

Univ. of California Santa Barbara
Santa Barbara, CA, USA
xavier@create.ucsb.edu

Resum

En aquest article presentem CLAM, un framework en C++, guardonat internacionalment, que ofereix una completa caixa d'eines per fer recerca i desenvolupament en el domini de l'àudio i la música. CLAM ofereix un model abstracte per sistemes multimèdia i inclou un repositori d'algoritmes de processat i tipus de dades, així com totes les eines necessàries per l'entrada i sortida d'àudio i control. Adicionalment, l'entorn conté una sèrie d'aplicacions llestes-per-usar per realitzar tasques com anàlisis i síntesis espectral de l'àudio, desenvolupament de plugins a temps-real i extracció de descriptors i anotació de meta-dades. L'article es centra, però, en les eines de prototipatge ràpid que permeten explotar totes aquestes funcionalitats a base de construir aplicacions eficients i multi-plataforma de forma totalment visual sense necessitat de programar.

1 Introducció

CLAM és un framework (que podria traduir-se aproximadament per “entorn de treball”) en l'àmbit de l'àudio i la música. Un framework és més que una aplicació informàtica, és un entorn complert que permet de crear aplicacions diverses en un camp concret, com ara el de l'àudio. Per a facilitar la creacions de noves aplicacions un framework sol oferir un model abstracte i una sèrie d'eines i components que es poden re-utilitzar directament.

La història dels frameworks està molt relacionada a amb l'evolució del camp de la multimèdia. Molts dels exemples d'entorns de programació més exitosos i coneguts tenen a veure

amb gràfics, imatge i multimèdia¹. Malgrat que no és tant conegut, el camp de l'àudio i la música també té una llarga tradició d'eines de desenvolupament similars. Exemples d'aquests entorns són: PD [13], Marsyas [14], Open Sound World [9]. I és en aquest context on trobem CLAM, un entorn de programació desenvolupat principalment a la Universitat Pompeu Fabra i que recentment ha rebut el premi Millor Software Multimèdia de Codi Obert del 2006, atorgat per la prestigiosa associació ACM.

CLAM és un acrònim de *C++ Library for Audio and Music*. El nom, però, es va escollir perquè el significat de la paraula “clam” en català està en certa manera lligat al món del so.

El framework permet crear aplicacions complertes i eficients en el domini de l'àudio i de la música que, a més, són totalment portables, funcionant amb el mateix aspecte a les plataformes GNU/Linux, MS Windows i Mac OSX.

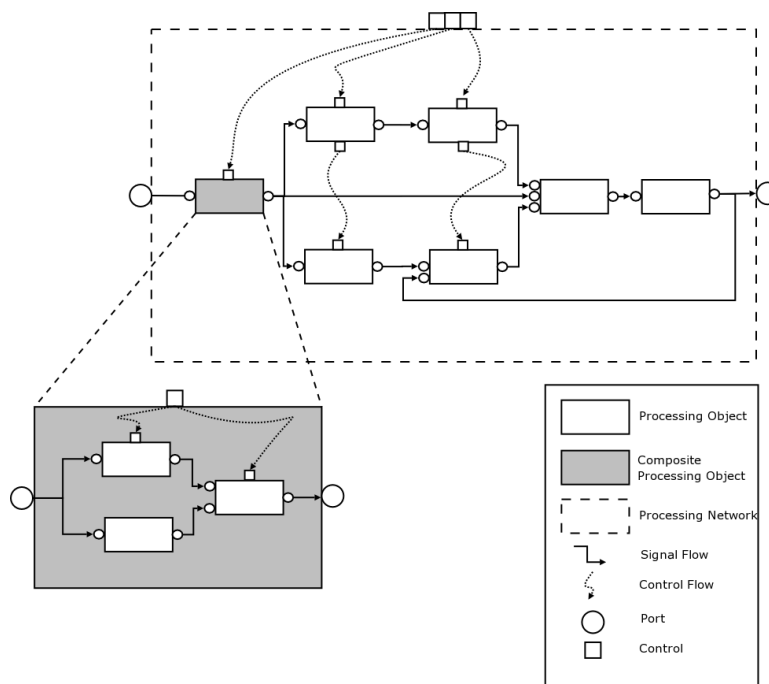


Figura 1: Metamodel de CLAM: consta d'una xarxa d'elements de processat connectats entre sí per ports i controls. Els ports porten el flux-de-dades, que és de natura síncrona o periòdica. Els controls porten events que són de natura esporàdica o imprevisible. Per tenir major expressivitat, les xarxes són composables recursivament.

La metodologia desenvolupament usada a CLAM incorpora molts elements de metodologies àgils i de desenvolupament obert (FOSS). Per exemple, usem *Desenvolupament Dirigit per Tests* i constants *Refactorings*, així com eines de testeig automàtic. Com a eines de testeig usem el framework CppUnit per testejar el codi C++ i una aplicació client-servidor, que ha estat desenvolupada a mida per les necessitats de CLAM anomenada TestFarm [8] —i que ha resultat també útil per altres projectes ben diferents. El funcionament d'aquesta eina

¹El primer entorn de programació orientat a objectes considerat com a tal són el MVC (Model View Controller) per Smalltalk[11] i el MacApp per les aplicacions d'Apple[15].

distribuïda és el següent: a cada integració de codi (*commit*) al repositori es desperten una sèrie de clients remots en diferents plataformes (Linux, Mac...) que es posen a compilar tot el codi i a executar suites de tests automàtics. El resultat es monitoritza via web, com es pot veure a la figura 2. Aquesta forma de treballar potencia la integració freqüent dels canvis i permet mantenir el codi estable en totes les plataformes malgrat que es treballi des d'una única plataforma —que típicament és Linux. I, finalment, evita trencar el ritme de desenvolupament ja que no cal esperar el resultats de les compilacions i els (llargs) tests automàtics: és TestFarm qui “desperta” al desenvolupador quan alguna cosa s’ha trencat.

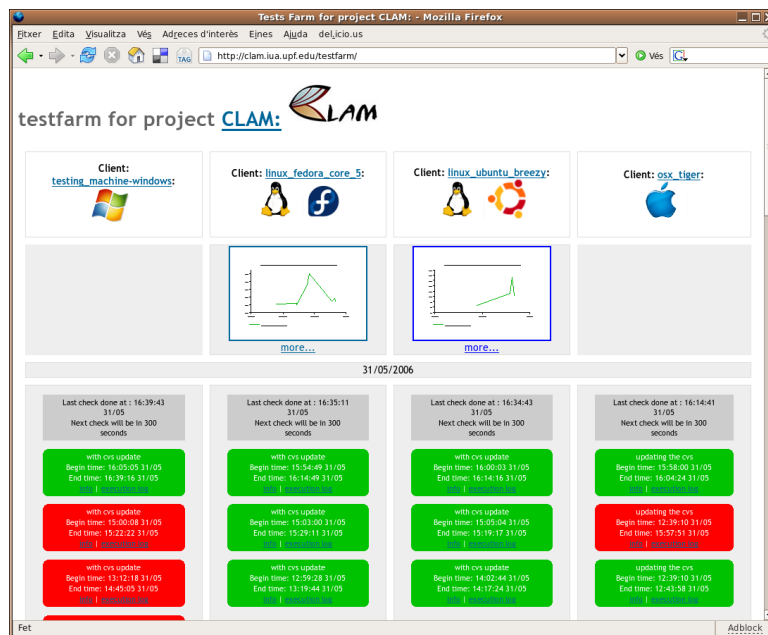


Figura 2: Monitorització web del Testfarm. Les columnes corresponen a clients remots en diferents plataformes. Les caixetes verdes indiquen una versió de codi que ha compilat i passat tots els tests, les vermelles indiquen algun tipus d’error que pot ser explorat desde la interfície web.

CLAM està llicenciat sota la GPL i s'utilitza, de forma creixent, principalment en el món de la recerca i l'educació però darrerament també en projectes comercials.

CLAM representa un pas endavant respecte altres entorns existents en el domini multimèdia però, alhora, també comparteix models i solucions de disseny amb altres entorns. Aquestes parts comunes poden ser expressades en forma de *metamodel* [2] per sistemes de processat multimèdia —la figura 1 en fa una breu pinzellada— i un *llenguatge de patrons de disseny*[7].

Malgrat que parts del framework ja han estat presentades en diversos articles (per exemple veure [3], [5], [6] i [4]) aquest article és únic perquè se centra en les novetats del l'entorn de construcció visual d'aplicacions. La següent secció entra de plè en aquest aspecte.

2 L'entorn de Construcció Visual d'aplicacions de CLAM

CLAM pot ser usat per desenvolupar aplicacions programàticament en C++. Però els últims desenvolupaments han incorporat la funcionalitat de *construcció visual* d'aplicacions (sovint també anomenat *prototipatge visual*). El principal avantatge és que permet a l'usuari concentrar-se en el desenvolupament d'algorismes de processat i no en el desenvolupament d'aplicacions. La construcció visual també és molt valuosa per fer prototips ràpids i eficients d'aplicacions i plugins d'àudio. No només permet una major productivitat sinó que amplia enormement la base d'usuaris ja que no es requereix coneixements de programació.

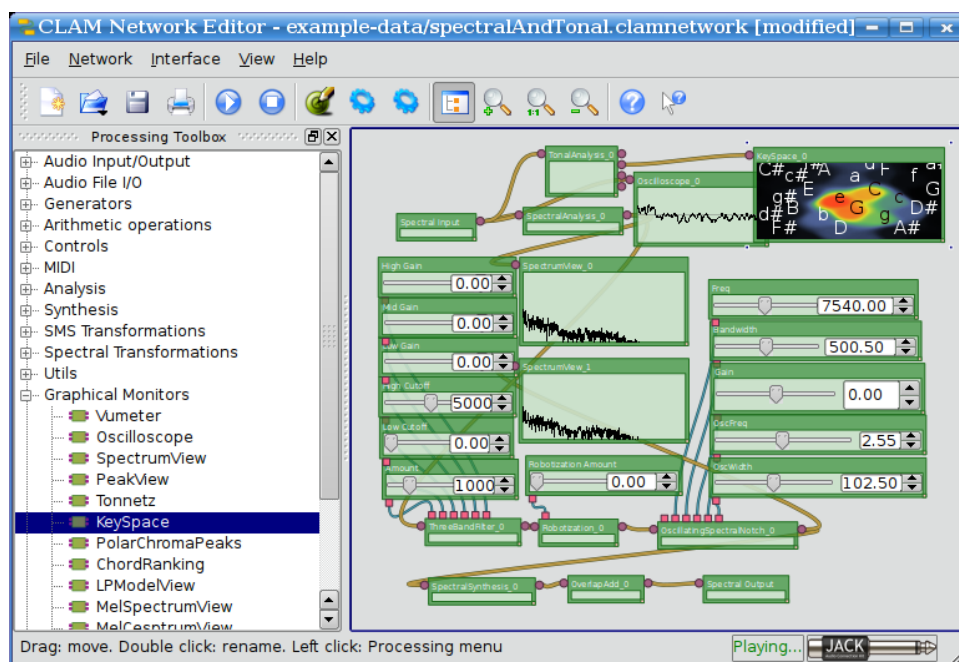


Figura 3: El Network Editor dóna accés visual al metamodel de CLAM. Podem arrossegar components de la biblioteca i connectar-los per construir un nucli de processat. Alguns components especials permeten enviar senyals de control i visualitzar dades de diferents tipus.

El constructor visual de CLAM s'anomena NetworkEditor (veure la figura 3). Permet generar un algorisme de processat a temps-real a base de connectar mòduls de processat en una xarxa de flux-de-dades. Una altra aplicació anomenada Prototyper fa de cola d'unió entre una eina de disseny d'Interfícies Gràfiques d'Usuari (com ara Qt Designer) i el sistema el giny de processat (també anomenat de flux-de-dades) definit amb el NetworkEditor.

És important subratllar que les aplicacions prototipades amb CLAM no tenen cap perjudici en eficiència respecte a les seves equivalents programades manualment. Això és així perquè enlloc hi intervé codi interpretat. Tot el que es fa és definir, construir i connectar estructures d'objectes en temps d'execució. L'aplicació resultant, per tant, executa únicament codi compilat, programat en C++.

Actualment existeixen moltes eines de prototipatge visual d'interfícies gràfiques. Exemples d'aquestes eines disponibles com a programari lliure són Qt Designer, Fltk Fluid o Gtk Glade. Però aquestes eines només resolten la composició de components gràfics en un *layout*, oferint una reactivitat limitada a la interacció de l'usuari; no serveixen per evitar la programació necessària per resoldre problemes típics que una aplicació d'àudio ha d'afrontar. Aquests problemes principalment tenen a veure amb la comunicació entre el giny de processat i la interfície d'usuari.

Aquesta secció descriu breument l'arquitectura de CLAM que soluciona aquest problema i habilita la construcció totalment visual d'aplicacions d'àudio a temps-real a base de combinar eines de disseny visual de flux-de-dades i eines de disseny visual d'interfícies gràfiques d'usuari.

2.1 Aplicacions realitzables

El conjunt d'aplicacions que l'arquitectura permet construir visualment inclou aplicacions de processat d'àudio a temps-real com sintetitzadors, analitzadors de característiques de l'àudio i plugins d'efectes d'àudio (exemples de protocols de plugins són: VST, AudioUnit i LADSPA).

L'única limitació imposada per l'arquitectura, sobre les aplicacions realitzables és que la seva lògica d'aplicació es limiti a engegar i parar l'algorisme de processat, configurar-lo, connectar amb el *back-end* d'àudio (dispositius d'àudio, hosts de plugins, MIDI, fitxers, OSC...), visualitzar les dades internes (per exemple amb una vista tipus oscil·loscopi) i controlar paràmetres de l'algorisme mentre aquest s'executa. De fet, aquestes limitacions tenen molt a veure amb el cicle de vida explicat en el meta-model de CLAM [2].

Una part important del programari d'àudio existent s'escau dintre d'aquestes limitacions. Altre tipus d'aplicacions d'àudio cauen fora del que permet fer l'arquitectura. Per exemple, eines d'edició no lineal d'àudio o aplicacions amb una lògica d'aplicació més complexa. Però, l'arquitectura és útil, fins i tot, per construir parts importants d'aquest grup d'aplicacions més complexes.

Aquestes són les principals funcionalitats que ofereix l'arquitectura:

- Comunicació de qualsevol tipus de dades (no només buffers d'àudio) i events de control entre la GUI i el giny de processat.
- El prototip pot ser incrustat dins una aplicació més àmplia amb un esforç mínim.
- Extensibilitat de l'arquitectura via plugins. Es poden estendre els elements gràfics de visualització de dades i d'enviament de controls, els elements de processat, i els back-ends d'àudio del sistema. (JACK, ALSA, PORTAUDIO, LADSPA, VST, AudioUnit...).

2.2 L'arquitectura

L'arquitectura del Construcció Visual (figura 4) consta de tres components principals. Una eina visual per definir el giny de processament d'àudio, una eina visual per definir la interfície gràfica d'usuari i un tercer element, el giny de run-time, que construeix de forma dinàmica

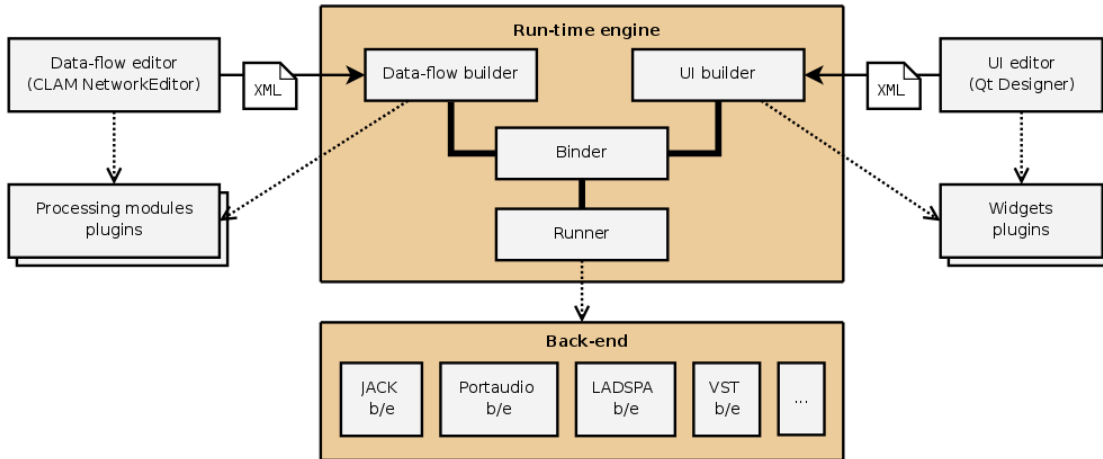


Figura 4: Arquitectura del Constructor Visual d'Aplicacions. El mòdul central és el giny de run-time, que usa plugins de components de processat d'àudio i plugins de components visuals (*widgets*) així com un back-end d'àudio d'entre tots els disponibles.

les definicions provinents dels dos àmbits (interfície i giny d'àudio), afegeix vincles entre els dos i manega la lògica d'aplicació. A CLAM, hem implementat aquesta arquitectura utilitzant eines existents: el NetworkEditor de CLAM és el constructor visual pel giny d'àudio, i el Qt Designer de Trolltech és el constructor visual de la interfície d'usuari. És interessant veure com totes dues eines —NetworkEditor i Qt Designer— ofereixen capacitats similars en el seu propi domini. Posteriorment, aquestes són explotades en el giny de run-time.

El Qt Designer és usat per definir interfícies d'usuari a base de combinar diferents components visuals (*widgets*). El conjunt de components visuals disponibles no està limitat; els desenvolupadors en poden definir de nous i afegir-los a l'eina visual com a plugins. La figura 5 mostra una sessió de Qt Designer on s'està dissenyant una interfície per una aplicació d'àudio usant un plugin de components visuals de CLAM. Es pot veure com els components visuals relacionats amb CLAM estan disponibles al panell de mà esquerra. Hi ha, per exemple, un component visual per l'àudio, un pels pics espectrals, un pels descriptors tonals (acords), etc.

Les definicions de interfície es guarden en fitxers XML amb la extensió “.ui”. I les aplicacions poden instanciar i descobrir la seva estructura en temps d'execució gràcies a les seves capacitats de introspecció.

De forma anàloga, el NetworkEditor de CLAM permet combinar diferents mòduls de processat de forma visual formant una definició de xarxa de processament. El conjunt de mòduls de processament de CLAM també és extensible amb llibreries endollables. Les definicions de xarxes de processament també poden ser guardades en fitxers XML que, més tard, poden ser carregats en temps d'execució per una aplicació. Finalment, CLAM també proveeix la capacitat de introspecció, de manera que una aplicació carregadora pot descobrir l'estructura d'una xarxa que s'ha carregat en temps d'execució. La figura 6 mostra de forma gràfica aquest funcionament.

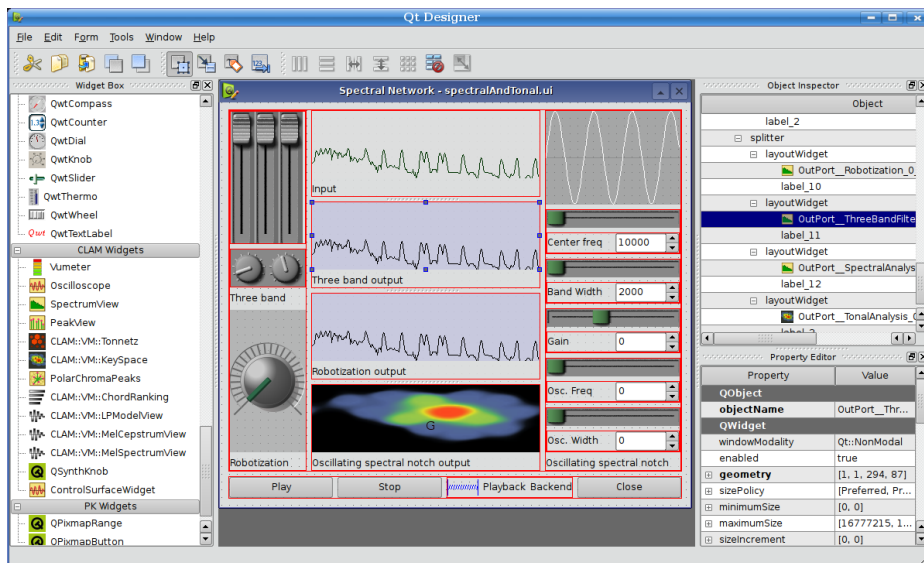


Figura 5: El Qt Designer es fa servir per construir interfícies composant diferents elements gràfics interactius (widgets). Podem controlar la composició, les propietats dels components i una interacció bàsica entre ells. CLAM hi afegeix nous components gràfics especials per àudio.

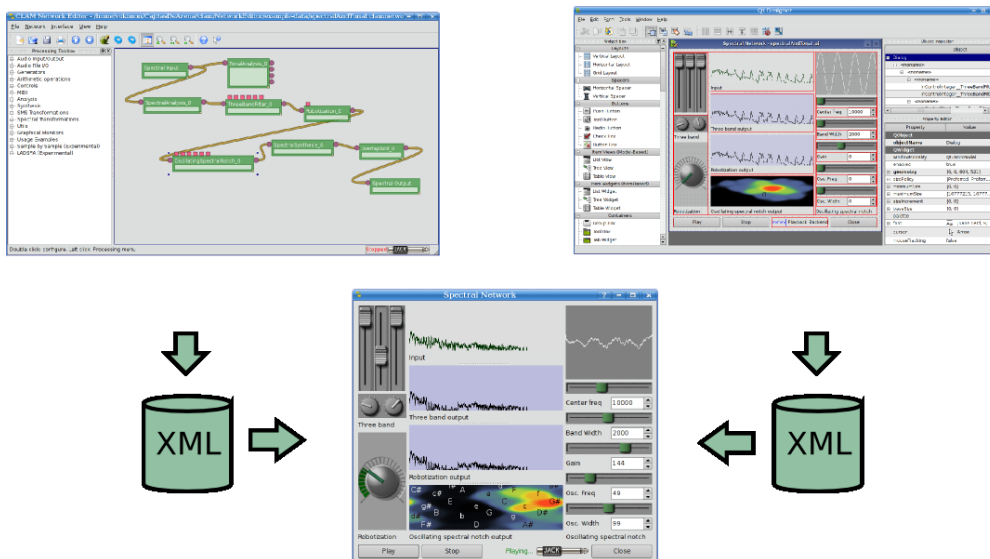


Figura 6: El flux de treball: Es dissenya el giny de processat (flux-de-dades) amb el NetorkEditor i la interfície d'usuari amb el QtDesigner, tot relacionant ports i controls a les propietats del Designer. A continuació el Prototyper de CLAM carrega ambdues definicions de fitxers XML i posa en marxa l'aplicació.

2.3 El giny de run-time

Si només disposéssim de les dues eines anteriors —una per dissenyar el flux-de-dades del giny d'àudio i l'altra per la interfície d'usuari— encara faria falta programar per crear una aplicació que uneixi les dues parts. El propòsit del giny de run-time, el qual anomenem Prototyper a CLAM, és proveir aquesta cola d'unió de forma automàtica. A continuació enumerem els problemes de disseny que es plantegen i com els solucionem.

2.3.1 La construcció dinàmica

Tant l'estructura d'objectes del giny d'àudio com la de la interfície gràfica d'usuari han de ser construïdes dinàmicament en temps d'execució a partir d'una definició XML. Tots dos frameworks, CLAM i Qt, suporten la construcció dinàmica ja que proveeixen *factories*[10][1] que instancien objectes donat un identificador del tipus.

Per aconseguir que els components de processat com els de interfície siguin expandibles, les factories han de suportar incorporar nous objectes definits en llibreries endollables. A tal efecte, es registren les classes a les factories en temps de inicialització del plugin.

2.3.2 Connectant objectes

La interfície d'usuari necessita enviar controls al giny d'àudio i, alhora, des de la interfície s'ha de poder visualitzar les dades que flueixen pel giny d'àudio. Per tant cal connectar adequadament objectes dels dos àmbits. L'arquitectura proposada usa propietats dels components visuals i els components de processat, com ara el nom del component (veure la figura 7)

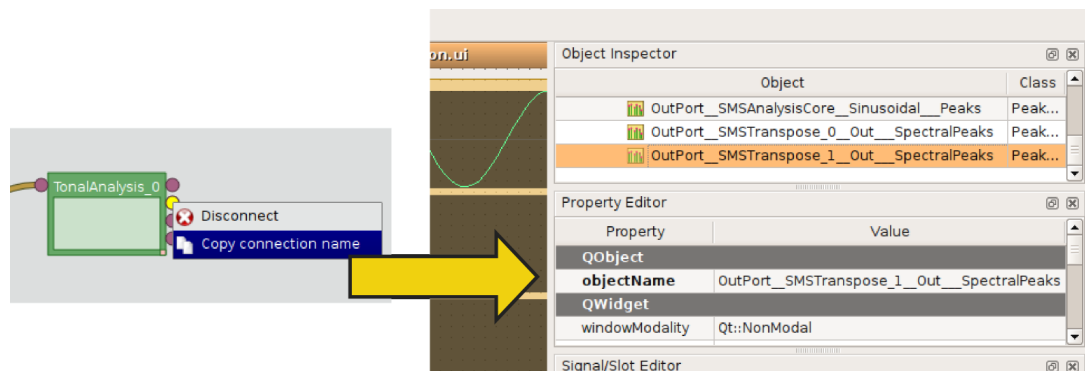


Figura 7: Connexió d'elements gràfics del designer amb ports i controls dels objectes de processament de CLAM

Els components (o sub-components) són localitzats usant les capacitats d'introspecció que ofereix cada un dels frameworks.

Un cop localitzats, el giny de run-time s'assegura que els components són compatibles entre ells. Com que no és possible que el giny conegui els tipus de dades que els objectes a connectar manegaran, apliquem el patró *Typed Connections* [7]. En breu, aquest patró de disseny permet

establir una connexió tipada (en sentit de tipus C++) entre dos components sense que l'entitat que estableix la connexió conegui els tipus i alhora sigui de tipus segur (type-safe).

2.3.3 Comunicació thread-safe en temps-real

En programació de sistemes de flux-de-dades a temps-real, l'enginy de processat s'executa en un thread d'alta prioritat mentre que la resta de l'aplicació —la interfície d'usuari, per exemple— s'executa en un thread de prioritat normal, tal com indica el patró *Out-of-band i In-band partition* [12].

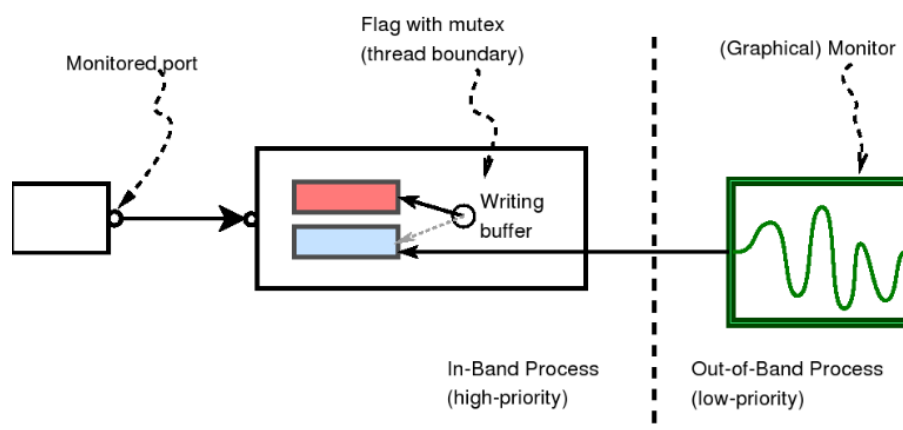


Figura 8: El patró *Port Monitor* —que forma part d'una col·lecció[7] de patrons relacionats amb CLAM— soluciona el problema de la comunicació entre dos threads, un d'ells d'alta prioritat. Fa servir tècniques *lock-free* i usa dos buffers i un flag que fa de barrera entre threads. El punt clau és que el thread d'alta prioritat consulta el flag sense mai bloquejar-se i sempre té un buffer disponible per escriure.

Com que es necessita comunicació entre threads, cal fer servir mecanismes de comunicació thread-safe, però els mecanismes tradicionals són bloquejants i les restriccions associades al temps-real impedeixen bloquejar el thread de processament. Per tant, cal usar tècniques no bloquejants, com les que hem proposat en el patró de disseny *Port Monitor*[7]. Veure la figura 8 per més informació sobre el patró.

2.3.4 Back-ends d'àudio

Les aplicacions d'àudio, per ser d'alguna utilitat, necessiten comunicar-se amb el món exterior. Hi ha molts possibles back-ends o arquitectures d'àudio. Per exemple, a Linux trobem ALSA, a Mac CoreAudio, a Windows ASIO. A més, les arquitectures de plugins d'àudio com ara VST, RTAS, AudioUnit o LADSPA poden ser considerades uns altres back-ends d'àudio quant a la programació d'aplicacions es refereix. La solució que l'arquitectura proposa per afrontar aquesta variabilitat, consisteix en tenir uns elements font (*sources*) i destí (*sinks*) a la xarxa de processat que són neutres quant al back-end d'àudio. I és el propi giny del run-time qui connecta les fonts i destí amb el back-end. Finalment, es disposen de plugins de back-ends

que adapten les variacions de cada back-end (la forma de fer threading, callbacks, assignació d'entrades i sortides) a una interfície comuna que a CLAM s'anomena NetworkPlayer.

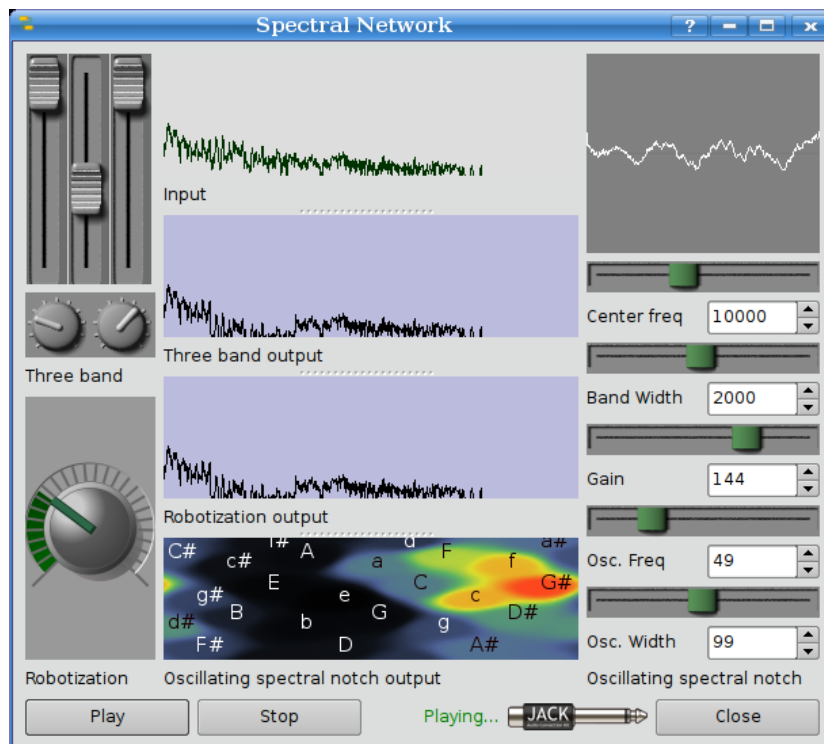


Figura 9: Exemple d'aplicació prototipada mentre s'executa usant el back-end d'àudio JACK.

2.4 Resultats de la construcció visual

En executar el Prototyper passant-li les descripcions XML de les eines visual n'obtenim una aplicació multiplataforma com la que surt a la figura 9. El nivell d'aprenentatge per tenir quelcom funcionant és molt baix. No cal perícia en programació; simplement un coneixement a nivell funcional dels components de processat. Fins i tot, l'eina de prototipatge és un medi ideal per aprendre i ensenyar els conceptes de processat.

Tot i que el sostre del que es pot fer sense programar es prou alt, la programació el permet aixecar encara més. Per un costat podem estendre el sistema amb nous components —sobretot visuals i de processat. La web de CLAM[16] conté tutorials curts (tipus *how-to*) sobre com programar nous components de processat i visualització i que ja han permès a usuaris amb mínims coneixements de programació contribuir nous components a la comunitat.

D'altra banda, si volem fer una aplicació amb una lògica més complicada, l'arquitectura encara ofereix al programador la possibilitat de fer servir el prototipatge per construir el nucli de processat i visualització, i incrustar-lo dins una aplicació més gran.

3 Conclusions

L'arquitectura de construcció visual d'aplicacions d'àudio presentada ja es troba totalment implementada en programari lliure dins el framework CLAM. El qual es troba disponible a la web de CLAM [16] per ser baixat i instal·lat en forma de codi o de instal·lables en diverses plataformes. De fet, junt amb CLAM distribuïm uns quants exemples d'aplicacions prototipades d'aquesta manera, com les mostrades en captures de pantalla en aquest article.

Amb aquesta funcionalitat un usuari pot construir aplicacions potents i eficients en pocs minuts, traient tot el profit a la llibreria de mòduls de processat. No menys important, els programadors es poden concentrar en el desenvolupament de components nous, evitant les qüestions complicades de programar una aplicació de temps-real.

Les idees de construcció visual d'aplicacions implementades en CLAM són innovadores i implementades recentment. Per tant, és natural que hi hagi espai per moltes millores. L'àrea que necessita més treball és la implementació de nous tipus de connexions per suportar components visuals de control complexes. També falta fer que els back-ends d'àudio funcionin realment com a llibreries plugins.

4 Agraïments

Els autors de l'article volen agrair el treball d'altres desenvolupadors que han intervingut en el desenvolupament del framework en el passat: Ismael Mosquera, Maarten de Boer, Jordi Massaguer, Xavier Oliver, Xavier Rubio, i Enrique Robledo. Així com els nous desenvolupadors actuals d'arreu del món: Hernan Ordiales, Andreas Calvo, Greg Kellum, Ebrahim Kazemzadeh, Bennet Kolasinski, Roman Goj i Zach Welch.

Volem agrair especialment el Ralph Johnson, un dels Gang of Four, qui ens va donar molta ajuda i ànims per millorar els nostres patrons de fluxe-de-dades, durant els workshops del PLoP. Agraïm també les contribucions dels experts en processament del senyal del Grup de Tecnologia Musical de la UPF. Josep Blat, de la UPF, ha ajudat en gran mesura donant suport al projecte en temps difícils. Donem gràcies a Google que ha concedit darrerament 6 beques Summer of Code a estudiants perquè treballin amb CLAM. El projecte ha estat parcialment finançat per una ajuda de la STSI de la Generalitat de Catalunya. Finalment, estem en deute amb l'equip de desenvolupadors de Trolltech i la Linux Audio Community per donar-nos la oportunitat de desenvolupar a partir de la seva feina.

Referències

- [1] ALEXANDRESCU, A. *Modern C++ Design*. Addison-Wesley, Pearson Education, 2001.
- [2] AMATRIAIN, X. *An Object-Oriented Metamodel for Digital Signal Processing*. PhD thesis, Universitat Pompeu Fabra, 2004.

- [3] AMATRIAIN, X. Clam: A framework for audio and music application development. *IEEE Software* 24, 1 (Jan/Feb 2007), 82–85.
- [4] AMATRIAIN, X., AND ARUMI, P. Developing cross-platform audio and music applications with the clam framework. In *Proceedings of the 2005 International Computer Music Conferenc (ICMC'05)* (2005). in press.
- [5] AMATRIAIN, X., DE BOER, M., ROBLEDO, E., AND GARCIA, D. CLAM: An OO Framework for Developing Audio and Music Applications. In *Proceedings of the 2002 Conference on Object Oriented Programming, Systems and Application (OOPSLA 2002)(Companion Material)* (Seattle, USA, 2002), ACM.
- [6] AMATRIAIN, X., MASSAGUER, J., GARCIA, D., AND MOSQUERA, I. The clam annotator: A cross-platform audio descriptors editing tool. In *Proceedings of 6th International Conference on Music Information Retrieval* (London, UK, 2005).
- [7] ARUMÍ, P., GARCIA, D., AND AMATRIAN, X. A data-flow pattern catalog for sound and music computing. In *Pattern Language of Programming PLoP 2006* (Oct. 2006).
- [8] ARUMÍ, P., SORDO, M., GARCIA, D., AND AMATRIAN, X. Testfarm, una eina per millorar el desenvolupament del programari lliure. In *Proceedings of V Jornades de Programari Lliure 2006; Barcelona* (July 2006).
- [9] CHAUDHARY, A., FREED, A., AND WRIGHT, M. An Open Architecture for Real-Time Audio Processing Software. In *Proceedings of the Audio Engineering Society 107th Convention* (1999).
- [10] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [11] KRASNER, G. E., AND POPE, S. T. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming* (1988).
- [12] MANOLESCU, D. A. A Dataflow Pattern Language. In *Proceedings of the 4th Pattern Languages of Programming Conference* (1997).
- [13] PUCKETTE, M. Pure Data: Another Integrated Computer Music Environment. In *Proceedings of the Second Intercollege Computer Music Concerts* (Tachikawa, 1996), pp. 37–41.
- [14] TZANETAKIS, G., AND COOK, P. *Audio Information Retrieval using Marsyas*. Kluewe Academic Publisher, 2002.
- [15] WILSON, D. A. *Programming With Macapp*. Addison-Wesley, 1990.
- [16] CLAM website: <http://clam.iuf.upf.edu>.