Un sistema de control de fluxe per síntesi i processament d'àudio en temps real

Enginyeria Superior en Informàtica

Universitat Pompeu Fabra

Xavier Rubio Campillo
xrubio@iua.upf.es

Director: Xavier Amatriain Rubio
xamat@iua.upf.es

16 de juny de 2003

Llicència d'aquest document

Permís garantit per a copiar, distribuir i/o modificar aquest document sota les condicions marcades per la **GNU Free Documentation License**, Versió 1.1 o posterior publicada per la **Free Software Foundation**.

Una còpia de la llicència està disponible a [16].

Abstract

L'objectiu d'aquest projecte final de carrera és aconseguir crear un mode automàtic de funcionament d'un sistema dissenyat per a processament de so i música. D'aquesta manera es preten que els usuaris de les aplicacions creades mitjançant aquest framework de desenvolupament puguin explorar les possibilitats sonores dels seus programes sense haver de preocupar-se de la gestió de dades entre els diferents elements que puguin fer servir.

En concret, el projecte es durà a terme amb un cas d'ús específic com a objectiu. Aquest és adaptar el mode automàtic per a una aplicació en temps real, l'instrument reacTable*.

Resum

El projecte consisteix en dissenyar un mode supervisat per a un framework de desenvolupament d'aplicacions relacionades amb el processament de so anomenat CLAM (*C++ Library for Audio and Music*) [14], desenvolupat pel Grup de Tecnologia Musical [35], que forma part de l'Institut de l'Audiovisual a la Universitat Pompeu Fabra.

CLAM, que actualment funciona de manera manual (és a dir, l'usuari ha de gestionar la comunicació entre els diversos elements del sistema), està pensat ja des de l'inici per tenir un doble funcionament; per una banda el mode llibreria que acabem de comentar, i per l'altra un mode aplicació en què l'usuari no s'hagi de preocupar de com van circulant les dades pel seu programa. Aquest és precisament l'objectiu del projecte.

A més, s'ha establert que aquest mode automàtic de CLAM haurà de servir com a *engine* de so d'un altre projecte del MTG anomenat reacTable*.

ReacTable*, desenvolupat pel grup d'interactius encapçalat per Sergi Jordà, pretén crear un instrument que sigui a la mateixa vegada complexe i intuïtiu. A més, s'està dissenyant per a funcionar en temps real. Per tant, és un cas d'ús molt representatiu per a fixar-lo com a primer objectiu a assolir pel mode supervisat de CLAM.

A més, hi ha un altre objectiu pel projecte, relacionat amb la creació d'una interfície gràfica. En els requeriments es demana algun tipus d'aplicació que, tot fent servir el mode automàtic de CLAM permeti als usuaris crear i modificar camins de processament de senyals de so sense haver de *barallar-se* amb la complexitat que pugui comportar un sistema amb diversos tipus de dades i que hagi de mantenir un alt grau d'eficiència. D'aquesta manera els dissenyadors de reacTable* podran provar nous algorismes de processat de manera senzilla i sense complicar-se amb el funcionament intern del framewoerk.

Índex

Taula de Continguts **10**

Índex de figures **12**

1 Introducció **13**

2 Glossari de termes **21**

3 L'anàlisi dels Requeriments **23**

3.1 Especificacions generals 24

3.1.1 Usuaris del sistema 26

3.2 Requeriments del sistema 26

3.2.1 Requeriments funcionals 26

3.2.2 Requeriments no funcionals 32

3.2.3 Requeriments del domini 33

4 Eines Requerides **37**

4.1 Paradigma de programació 38

4.1.1 Trets diferencials entre paradigmes 39

4.1.2 Elecció 39
4.1.3 Concurrència 40
4.2 Llenguatge de programació 40
4.2.1 Avantatges del C++ 41
4.2.2 Desavantatges 42
4.2.3 Llenguatges interpretats 43
4.3 El so 43
4.3.1 Requeriments per la llibreria de so 43
4.4 L'entorn gràfic 45
4.4.1 Elecció del toolkit gràfic 46
4.5 Eines de desenvolupament 46
4.5.1 Compilador 47
4.5.2 Editor de codi 47
4.5.3 Sistema Operatiu 47
4.5.4 Gràfics i diagrames 48
4.5.5 Sistema de control de versions 48
4.5.6 Eines de testeig automàtic 49

5 El model del sistema 51

5.1 Música dins un ordinador? 52
5.1.1 Paradigmes de programari musical 52
5.1.2 La idea dins reacTable* 57
5.2 El control de fluxe 58
5.2.1 Models síncrons i asíncrons 58

5.2.2 Fluxe de control vs fluxe de dades 61
5.3 Avaluació de l'ordre d'execució 62
5.3.1 MAX 63
5.3.2 Schedulers amb models amb prioritats 64
5.3.3 Sistemes basats en topologies de graf 64
5.4 CLAM: C++ Library for Audio and Music 65
5.4.1 Dynamic Types 66
5.4.2 Processing Objects 67
5.4.3 Processing Data 68
5.4.4 Modes de funcionament 68
5.4.5 Ports i controls 68
5.4.6 Nodes 69
5.4.7 Xarxa 70
5.4.8 Sistema complert 71

6 Disseny del sistema 73

6.1 Metodologies de desenvolupament: una visió teòrica 74
6.2 Aplicació de la metodologia al projecte 75
6.3 El camí fins al mode supervisat de CLAM 77
6.4 Nodes 78
6.4.1 La primera capa: la interfície dels nodes 79
6.4.2 Segona capa: Les regions 80
6.4.3 Tercera capa: La implementació de la cua 84
6.5 La xarxa de processament 87

6.5.1 Propietat dels objectes de processament 88
6.5.2 Comunicació amb l'algorisme de control d'execució 88
6.5.3 Serialització 89
6.6 Ordres d'execució 91
6.6.1 Algorisme d'exemple 91
6.7 Visualització 93
6.7.1 El mòdul de visualització de CLAM 93
6.7.2 Disseny de l'aplicació gràfica 96
6.7.3 Implementació de la presentació 97
6.8 Resultats 97

7 Conclusions **99**
7.1 Pensaments sobre la feina feta 99
7.2 Línies futures de treball 100
7.3 Avaluacions personals 101

8 Agraïments **103**

Bibliografia **109**

Índex de figures

1.1 Mòduls del framework CLAM 14

1.2 Exemple de connexions 14

1.3 Una altra xarxa de connexions amb els mateixos elements 15

1.4 Captura de pantalla d'un prototipus de reacTable* 17

3.1 Exemple de xarxa d'objectes de processament en CLAM 25

3.2 Exemple d'una possible xarxa amb subxarxes 27

3.3 Diagrama de casos d'ús relatiu a la serialització de la xarxa 28

3.4 Diagrama de casos d'ús de modificacions de la xarxa 30

3.5 Canvi de la xarxa que comporta una alteració de l'ordre d'execució 31

3.6 Example de representació temporal del so 33

3.7 Example de representació espectral del so 34

5.1 Captura de pantalla d'Ardour, extreta de la seva pàgina web 53

5.2 Captura de pantalla de Rebirth, extreta de la seva pàgina web 54

5.3 Captura de pantalla de MAX/MSP, extreta de la seva pàgina web 56

5.4 Captura de pantalla de FMOL, extreta de la seva pàgina web 57

5.5 En aquesta xarxa l'avaluació de l'ordre d'execució seria senzill 62

5.6 Aquí ens trobem en una avaluació bastant més complexa 63
5.7 Estructura d'un object de processament 67
5.8 Estructura del elements que componen el control de fluxe 71

6.1 Seqüència d'intercanvi de dades en la primera capa de l'estructura de nodes 79
6.2 Diagrama de classes de la primera capa de l'estructura de nodes 80
6.3 Interior d'un node amb dues regions de lectura i una d'escriptura 81
6.4 Seqüència d'intercanvi de dades en la segona capa de l'estructura de nodes . 82
6.5 Diagrama de classes de la segona capa de l'estructura de nodes 83
6.6 Problema de la implementació de nodes amb búffers circulars 84
6.7 Solució amb l'ús d'una zona fantasma 85
6.8 Seqüència d'intercanvi de dades en la tercera capa de l'estructura de nodes 86
6.9 Diagrama de classes de la tercera capa de l'estructura de nodes 87
6.10 Patró de disseny estratègia usat a la Network 89
6.11 Patró de disseny Factory Method per a crear nous objectes de processament 90
6.12 Diagrama de classes del Patró de disseny Model-Vista-Controlador 94
6.13 Diagrama d'interaccions entre els elements del patró Model-Vista-Controlador 95
6.14 Diagrama de classes d'un controlador de Network 96
6.15 Diagrama de classes de l'adaptador d'un objecte de processament 97
6.16 Captura de pantalla de l'editor de xarxes 98

Capítol 1

Introducció

Abans de començar a parlar del meu projecte, cal explicar l'entorn en el que el desenvoluparé i la seva motivació a fi d'entendre els objectius del mateix.

El Grup de Tecnologia Musical està creant una llibreria multiplataforma anomenada CLAM amb la finalitat d'ajudar a desenvolupar aplicacions futures relacionades amb l'àudio i la música. A més, també forma part del projecte europeu AGNULA [1], que està creant una distribució del sistema operatiu GNU/Linux composta integrament per programari lliure. Per tant, CLAM està publicada amb llicència GPL (General Public License) [17].

CLAM és molt més que una llibreria; és un framework en el qual s'intenta integrar totes les eines de programació que poden necessitar les aplicacions desenvolupades per l'MTG. Per tant, tenim un sistema amb diversos mòduls, tal i com veiem a la figura 1.1

Engloba tant aspectes concret del processament d'àudio com d'altres de relacionats, tenint mòduls per a la visualització de dades, serialització de resultats, etc.

Un dels aspectes més importants en un framework per a crear aplicacions d'àudio com és

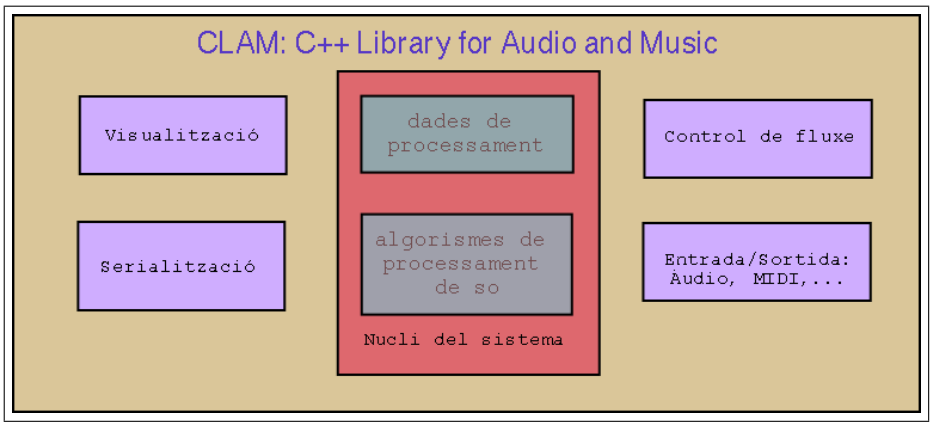

Figura 1.1: Mòduls del framework CLAM

CLAM és la manera en què es poden connectar els diferents algorismes de processament. Un exemple d'una col.lecció típica d'objectes de processament connectats seria la del diagrama 1.2.

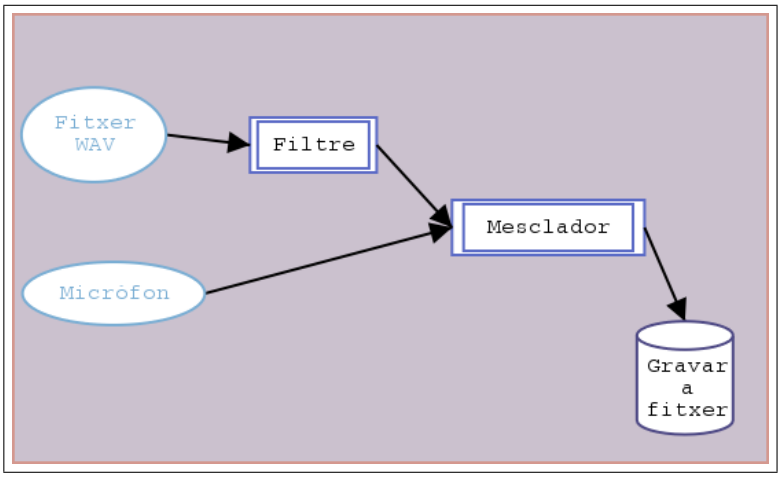

Figura 1.2: Exemple de connexions

La pregunta bàsica és: qui gestiona les dades que es passen els diferents mòduls connectats, i com s'executaran? En aquest sentit, CLAM té dos funcionaments diferents:

- Mode no supervisat

  Aquest és el mode natural de la llibreria, a on el programador ha de controlar ell mateix les connexions de dades (controls, mostres de so, etc.) de l'aplicació. Així, a més d'especificar quins algorismes s'utilitzaran, també s'ha de tenir en compte com es connecten i el camí (també anomenat fluxe) pel qual hauran de passar les dades

a fi de ser transformades tal i com es desitji. Ara per ara és l'únic implementat dins CLAM.

- Mode supervisat

  Aquest mode encara no està desenvolupat tot i que en les decisions de disseny de CLAM sempre s'han avaluat les seves conseqüències en el disseny. Amb aquesta mètode d'execució no faria falta que el creador d'aplicacions es preocupi de les connexions entre els diferents elements que formen un programa , ja que és el propi framework qui haurà de ser el responsable de respondre als canvis de connexions, tot modificant el sistema correctament.

  Per aclarir el paper que ha de fer aquest mode supervisat, podem pensar per exemple en un sistema amb diversos mòduls, cadascun dels quals transformi el so d'una manera determinada. Hi haurà moltes formes diferents d'enllaçar-los, cadascuna amb una sortida determinada i, a més, amb l'ordre en que s'aniran executant serà diferent en cada cas. El mode supervisat ha de permetre anar variant aquestes connexions i que tot el que se'n derivi sigui reajustat de manera automàtica. La figura 1.4 és un exemple del que estem comentant: els algorismes anteriors han estat connectats en una altra configuració del sistema. El sistema que s'encarrega de controlar aquests canvis en les connexions entre objectes de processament, i l'ordre en què s'han d'executar per tal d'interpretar-les correctament és l'anomenat control de fluxe, o *FlowControl*.

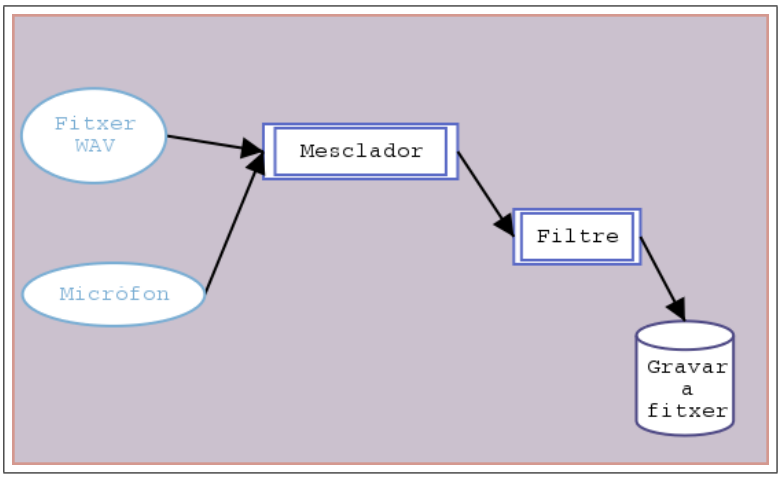

Figura 1.3: Una altra xarxa de connexions amb els mateixos elements

CLAM està essent dissenyada i implementada pels membres de l'equip de desenvolupament del MTG:

- Xavier Amatriain - Cap de projecte

- Pau Arumí
- Merlijn Blaauw
- Maarten de Boer
- David García
- Sandra Gilabert
- Albert Mora
- Miguel Ramírez
- Xavi Rubio

Per altra banda, el projecte reacTable* està encapçalat pel grup de sistemes interactius que ha creat l'MTG, Grup de Tecnologia Musical de l'Institut de l'Audiovisual a la Universitat Pompeu Fabra. Els seus components (els estudiants de doctorat Alvaro Barbosa, Gunter Geiger, Rubén Hinojosa, Martin Kaltenbrunner, José Lozano i el cap de projecte Sergi Jordà) pretenen crear un instrument que acumuli tota l'experiència en els diferents camps tecnològics que el MTG ha anat acumulant al llarg dels anys.

Tenint com a precedent el programa FMOL [26] (dissenyat per a l'obra Faust de la Fura dels Baus i usat posteriorment en concerts), es preten dissenyar un sistema musical que sigui intuïtiu i complex a la vegada. És a dir, ha de poder-se fer servir com a instal.lació per a què fins i tot les persones que no tinguin cap tipus d'estudis musicals puguin començar a produïr efectes sonors de manera inmediata, tot entenent la mecànica de l'instrument. Al mateix temps, ha de ser ric en quant a possibilitats, a fi que un músic pugui imaginar-se en poc temps quin és l'abast sonor de l'instrument. En poques paraules, reacTable* no ha de ser tan sols una *joguina* musical, sinó una nova forma d'interacció de l'home (sigui músic professional o no) amb el so i la música.

El sistema dissenyat per a assolir aquesta dualitat està basat en una taula amb diversos objectes dispersos per ella. El músic (o la persona que interactuï amb la taula) variarà el so produït movent els diferents objectes, ja que cada un d'ells simbolitzarà un objecte generador o modificador de so, que es connectarà a d'altres seguint unes regles predefinides (proximitat, per exemple). Podem veure en aquesta captura de pantalla l'estructura general de la taula i les seves connexions **??**.

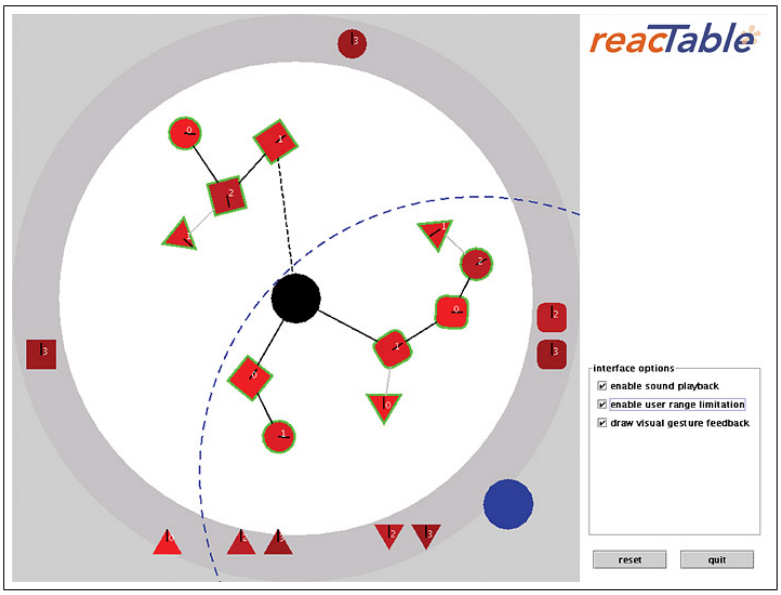

Figura 1.4: Captura de pantalla d'un prototipus de reacTable*

Com es veu, és un projecte realment sofisticat que farà servir tècniques tant visuals (amb projectors per a mostrar les connexions i reconeixement de formes per a detectar els objectes), com interactives i auditives (per motius obvis, ja que és un instrument musical).

Com veiem, CLAM és una llibreria òptima per al desenvolupament de la reacTable*, ja que permet tenir les eines necessàries per al processament sonor i la connexió als elements de visualització de manera que no s'hagi de partir de zero. A més, farà falta algun sistema semblant al mode supervisat per a controlar les connexions dels objectes sonors que hi haurà darrera dels objectes físics situats a la taula.

Això doncs, i pel que fa a la motivació, l'objectiu primari del meu projecte és crear el control de fluxe que necessita reacTable*, comprenent les relacions tant de dades d'àudio com de controls que es produiran en aquest sistema, i fer-ho de manera tal que s'integri perfectament amb el disseny d'aquest projecte.

A més, és important destacar el doble paper de CLAM i la seva relació amb aquesta tasca.

Per una banda, tenim el fet que el grup d'interactius ha decidit triar CLAM com a framework sobre el qual muntar l'aplicació. Per tant, en aquest sentit el disseny estarà lligat a la filosofia de disseny que s'ha donat a CLAM.

Per l'altra, podem veure que els requeriments del control de fluxe de reacTable* i del mode supervisat de CLAM són molt semblants. És més, realment el mode supervisat de CLAM que, com hem comentat abans, no està encara desenvolupat, haurà de respondre a les necessitats de projectes com el que ens ocupa. Per tant, es dissenyarà i implementarà el mode supervisat de CLAM, i posteriorment s'adaptarà a les necessitats concretes del projecte reacTable*.

Un segon objectiu del meu projecte final de carrera és dissenyar i implementar una interfície gràfica que permeti als membres del grup d'interactius desenvolupar nous objectes de processament i testejar-los de manera ràpida i còmode. És a dir, proporcionar una eina per a provar nous efectes sonors sense haver de fer servir la taula ni moure's de l'ordinador a on s'estigui implementant.

Ha de ser una aplicació senzilla però que, al mateix temps, sigui prou potent com per a emular adequadament quin serà el resultat de les diferents connexions entre objectes que es provin a la reacTable*.

Per tant, ara que he fixat a mode de petita introducció les motivacions i projectes als quals ha de donar resposta el projecte final de carrera, passarem a analitzar els requeriments que se m'han donat, així com les eines necessàries per a crear una solució adequada.

Capítol 2

Glossari de termes

Abans de fer un pas més endavant en l'anàlisi i disseny del projecte que ens ocupa, hem d'aclarir uns quants conceptes bàsics per tal de no caure en ambigüitats o errors de concepte.

Per tant, aquest és un petit glossari amb els termes [1] que s'aniran repetint al llarg de les diferents etapes de què consistirà el desenvolupament:

CLAM.- C++ Library for Audio and Music. Framework desenvolupat pel grup de Tecnologia Musical de l'Institut de L'audiovisual a la Universitat Pompeu Fabra. Està dedicat a proporcionar les eines necessàries per aconseguir desenvolupar de manera intuïtiva, completa i unificada aplicacions relatives al so amb tots els elements que això comporta ('Tractament d'entrada i sortida, MIDI, visualització, algorismes de generació, anàlisi i tractament de senyal temporal o espectral etc.

The reacTable*.- Projecte desenvolupat pel grup d'interactius del *MTG* consistent en dissenyar un instrument sonor complexe i a la vegada intuïtiu fent servir conjuntament tècniques de reconeixement de formes, projecció visual, algorismes de generació sonors i noves formes d'interacció home-màquina.

Processing object o Objecte de Processament.- Element del sistema que rep una sèrie de dades, les modifica seguint les directrius de comportament marcades per una configuració associada a ell i els controls que té connectats, i les reenvia a d'altres objectes de processament connectats a les seves sortides. Internament pot estar composat d'altres objectes de processament.

FlowControl o Control de fluxe.- Gestor que s'encarrega d'organitzar i dirigir tots els tipus de dades produïdes o demanades per un objecte de processament determinat del sistema.

[1] Molts termes s'empraran amb la seva denominació anglesa i catalana indistintament.

Fluxe de control o ControlFlow.- Aquella part de les dades de l'aplicació relativa a controls entre objectes, és a dir, a dades que modifiquin el comportament de l'element especificat, tot estudiant les relacions que s'estableixen dins l'etapa d'execució.

Fluxe de dades o DataFlow.- Estudi dels camins de les dades amb les què han de tractar els diferents objectes del sistema, i que forment part de la informació a tractar pròpiament dita per l'aplicació.

Port.- Dins la terminologia de CLAM, element pel qual entren les dades a un objecte de processament i, per tant, relacionat directament amb el fluxe de dades.

Control.- Dins la terminologia CLAM, element pel qual entren les dades relacionades amb la modificació del comportament d'un objecte de processament i, per tant, relacionat directament amb el fluxe de controls.

Node.- Element del FlowControl que es preocupa de coordinar la sortida d'un objecte de processament amb les entrades d'un o més Processing Objects que volen consumir les dades produïdes pel primer.

Network o Xarxa.- Conjunt de diversos objectes de processament connectats entre ells mitjançant ports i/o controls.

Ordre d'execució.- Jerarquia per la qual s'hauran d'anar aplicant els diferents objectes de processament tenint en compte les connexions entre ells.

Avaluació de l'ordre d'execució.- Tasca dessignada a un subsistema que s'encarrega d'anar executant els objectes de processament d'una xarxa en l'ordre correcte, tenint en compte les relacions entre ells.

Scheduler o planificador.- Tipus d'avaluació de l'ordre d'execució en què es decideix l'ordre d'execució tenint en compte el coneixement que es té del sistema i les accions necessàries per al correcte funcionament del sistema. Aquest planificador pot estar basat en prioritats, ordre jeràrquic, etc.

Mode no Supervisat.- Funcionament en mode *llibreria* de CLAM, segons el qual el dissenyador de l'aplicació implementa també el control de fluxe de forma manual.

Mode Supervisat.- Dins la terminologia CLAM, mode pel qual la planificació de l'execució dels diferents objectes de processament, a més de la relació entre ells, són controlats de manera automàtica per una secció de l'aplicació anomenada scheduler, no pel dissenyador.

Domini temporal de so.- Representacio normal de les dades d'àudio, com a mostres donades per l'amplitud i la fase de les freqüències escoltades.

Domini espectral de so.- Representació donada per l'aplicació de la transformada de Fourier a les mostres de domini temporal. El resultat és un anàlisi per freqüències del so analitzat.

Buffer.- Número de mostres (de domini temporal o espectral) que són processades juntes a cada execució d'un algorisme. Poden ser buffers temporals (per exemple de 256 mostres de so), o espectrals (per exemple 4 frames, cadascun d'ells comprenent l'anàlisi espectral d'una secció de temps determinada).

Capítol 3

L'anàlisi dels Requeriments

Un cop complerta una etapa introductòria a on s'han assenyalat quins són els objectius inicials del sistema, s'haurà de definir detalladament quins requeriments serà necessari fixar per tal d'ajustar-se a les necessitats del projecte, per tal de no caure en ambigüitats i errors de disseny produïts per una mala comprensió de la situació.

Primer de tot, s'exposaran de manera general quins són els objectius del sistema, i a quins problemes haurà de donar solució. Un cop fet això, serà necessari desenvolupar els requeriments del sistema. Aquests tenen la seva utilitat en la mesura que són una definició bastant completa i concreta sobre el que ha de fer i el que no ha de fer el sistema. És a dir, mentre que un resum inicial serveix per fer-se una idea global i en llenguatge bastant planer sobre el que ha d'aconseguir un projecte d'enginyeria del programari (i així poder-ho fer servir per situar-se en el context del sistema), l'aprofundiment d'aquestes idees inicials té el propòsit d'expandir aquests objectius fins a un punt molt més proper a l'enginyer, de manera que a partir d'ells es pugui arribar a un bon disseny del sistema.

Així aconseguirem que el projecte respongui als diferents problemes concrets que se'ns

plantegin, però sense perdre de vista els conceptes globals, que són fonamentals.

Dins aquests requeriments podem fer una divisió relativa a la font de la qual provenen. En tindrem de tres tipus:

- Requeriments funcionals
  Són els relatius al que ha de fer el sistema i a quines necessitats ha donar resposta.
- Requeriments no funcionals
  Aquests no són propis de les funcionalitats de l'aplicació, sinó que venen donats per altres factors (temps, eficiència,etc.). Normalment acostumen a ser més restrictius que no pas definitoris.
- Requeriments del domini
  Totes les característiques pròpies del domini al qual pertany el projecte, i que d'alguna manera poden afectar el disseny.

Aquesta divisió ens facilita molt la feina, ja que permet tenir sempre clar quins són els objectius primaris del sistema, quines condicions ha de complir el nostre disseny, i totes les peculiaritats del domini que s'han de tenir en compte alhora de crear l'aplicació.

D'altra banda, i enfocant els requeriments en relació amb el projecte que ens ocupa, cal comentar que, tenint en compte els objectius i motivacions del projecte, és molt adient que dins cada etapa s'intentin esbrinar quins són els requeriments comuns i els especials entre el mode supervisat de CLAM i el control de fluxe de reacTable*. És a dir, tal i com hem comentat a la introducció, aquest projecte aborda les dues vessants, tenint en compte l'estreta relació entre els dos projectes. Més concretament, la secció relativa a reacTable* és, com ja veurem en l'explicació detallada dels requeriments, un cas concret d'aplicació del mètode supervisat de CLAM. Tot i això, hi ha una série de requeriments propis que variaran el disseny de la proposta general representada per aquest sistema automatitzat de CLAM.

Així doncs, passarem ara a comentar els punts generals, per més tard desenvolupar-los adientment. Per fer una ordenació coherent, seguiren la classificació anterior.

3.1 Especificacions generals

Un primer objectiu del sistema és intentar definir el concepte de xarxa, entès dins el domini de l'àudio com un conjunt d'*instruments* que generen sons o els modifiquen, tenint en compte les dades proporcionades per una série de controls que són variats per una entitat externa (sigui un gestor de controls, un humà, un altre objecte, etc.) i la configuració

d'aquests instruments. Concretament parlant del framework de processament CLAM, els instruments són algorismes de processament de so encapsulats dins objectes de processament. Un exemple del que pot ser una xarxa és la figura 3.1

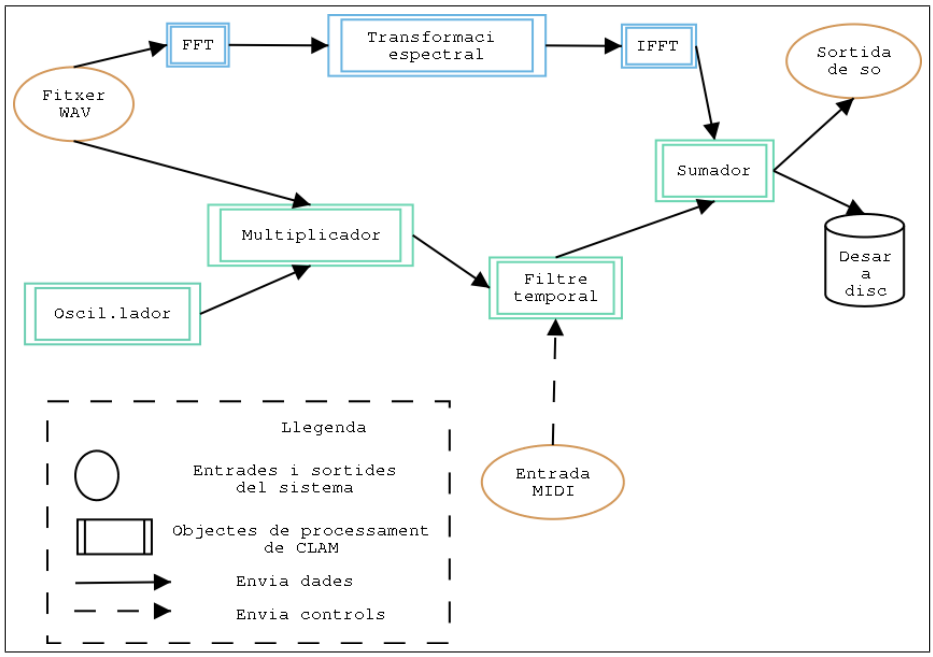

Figura 3.1: Exemple de xarxa d'objectes de processament en CLAM

Aquesta xarxa també haurà de ser capaç de saber en tot moment l'estat dels objectes de so que en formen part, a més de les connexions que hi hagi entre ells, que podran variar en temps d'execució.

Per tant, serà necessari algun mecanisme que permeti als objectes que necessitin informació sobre la xarxa aconseguir-la (quines connexions hi ha, els tipus de dades, etc.).

També es preten dissenyar un element automàtic que controli l'ordre d'execució dels objectes. És a dir, necessitem una entitat que, tot col.laborant estretament amb una xarxa, pugui mantenir el fluxe de dades entre objectes de processament de tal manera que no es produeixin retards o errades del sistema, decidint en quin ordre s'hauran d'executar donada la disposició de les connexions i objectes de processament (l'anomenada topologia de la xarxa).

Un altre requeriment bàsic serà el fet que la configuració del sistema i de la xarxa hauran de ser serialitzables en qualsevol moment, a fi que a l'engegar l'aplicació es pugui carregar una topologia concreta des de fitxer.

Pel que fa a l'aplicació gràfica, haurà de ser possible crear i modificar una xarxa, afegint o

traient connexions, objectes de processament, etc. Per tant, la idea bàsica és dissenyar un editor gràfic de xarxes de CLAM en temps real.

3.1.1 Usuaris del sistema

Aquest sistema no està orientat cap a enginyers ni programadors, sinó que serà usat per músics i fins i tot persones sense grans coneixement de música (com es va comentar a la introducció, haurà de fer-se servir en un doble paper d'instal.lació i intrument musical complex). Per tant, la gestió de connexions, nodes, i en definitiva, la organització de la xarxa, haurà de ser completament transparent a l'usuari, així com l'avaluació de l'ordre d'execució, que haurà de ser automàtica.

3.2 Requeriments del sistema

Un cop especificats els objectius de forma general, ara cal expandir aquestes premises bàsiques, tot expressant-les d'una forma més concreta i tècnica com a requeriments del sistema.

Com he comentat abans, es farà una divisió relativa a l'origen d'aquests.

3.2.1 Requeriments funcionals

He separat aquesta secció en tres blocs, tenint en compte els tres grans mòduls que formaran conjuntament el sistema: la xarxa, el control de fluxe i l'aplicació gràfica.

La xarxa

S'haurà de dissenyar una xarxa que pugui contenir els diferents objectes de processament (cadascun amb el seu estat corresponent), els nodes que continguin les dades processades, i les connexions entre objectes, ja siguin de control o ports.

Un requeriment del mode supervisat de CLAM és que aquesta xarxa ha de poder contenir, a la seva vegada, altres xarxes, de manera que enlloc de tenir una sola xarxa puguem tenir una estructura en forma d'arbre. Pel que fa a reacTable*, aquesta característica no és necessària, ja que a la taula no podem representar les subxarxes com a objectes físics de la mateixa manera que els objectes de processament, i per tant, en aquest context el concepte de subxarxa no té sentit.

Dins el mode supervisat de CLAM, aquesta subxarxa haurà de ser capaç de "publicar" un seguit de ports i controls. És a dir, vista desde fora aquesta subxarxa tindrà una interfície semblant a la d'un objecte de processament típic, amb una série de ports d'entrada i de sortida, i també una llista de controls (tant per a rebre dades com per enviar-ne). En el diagrama 3.2 podem veure la xarxa anterior, aquest cop amb una subxarxa que engloba els objectes de processament relacionats amb dades espectrals. A la seva vegada, les transformacions que es fan també podrien ser subxarxes.

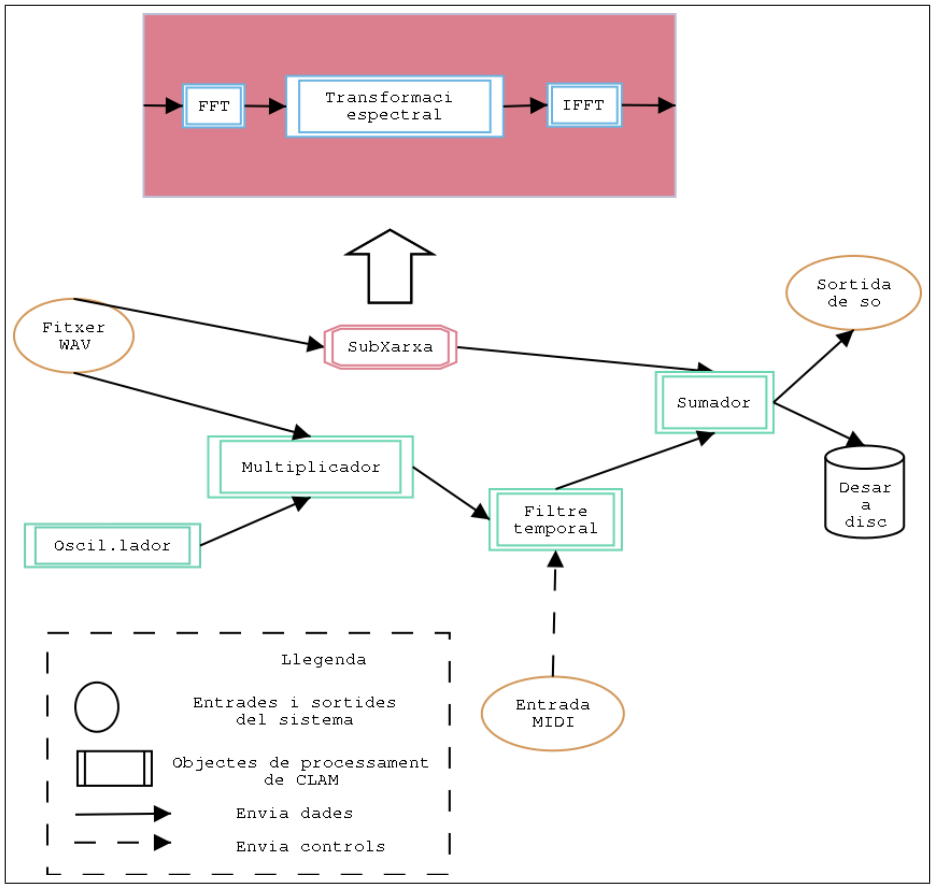

Figura 3.2: Exemple d'una possible xarxa amb subxarxes

Aquesta xarxa haurà de ser serialitzable, de tal manera que en qualsevol moment durant l'execució del sistema, podrem desar a disc la xarxa que s'està modificant. Això vol dir que s'haurà de passivar:

- Els Objectes de processament dels qual està composada la xarxa
- La configuració dels objectes de processament

- Les Connexions de dades entre els objectes
- Els enllaços de control entre aquests objectes

Aquesta serialització, seguint el model de CLAM, es farà en XML, de tal manera que l'usuari podrà, si ho desitja, editar aquesta configuració abans que sigui carregada al sistema. Com veiem al diagrama 3.3, les dues accions principals que podem dur a terme són desar l'estat de la xarxa, i recuperar-la en un moment posterior.

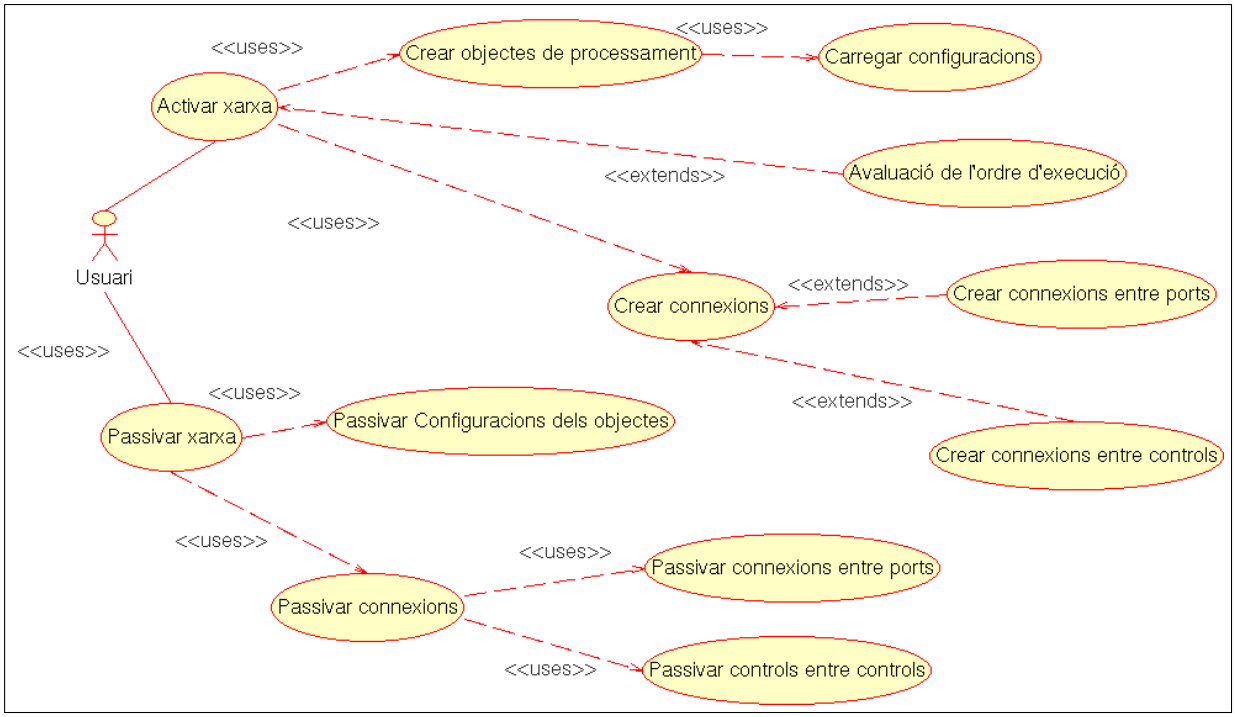

Figura 3.3: Diagrama de casos d'ús relatiu a la serialització de la xarxa

Al passivar s'hauran de guardar totes les connexions que hi ha entre objectes de processament, i per cada un d'ells, la seva configuració. Aquesta configuració conté dades relatives als paràmetres que caracteritzen l'objecte, com podria ser el nom, el número de mostres a processar a cada *firing* [1], o el tipus de processament a fer d'entre una selecció pròpia de l'objecte. Per exemple, en un generador de so haurem de triar la forma de l'ona generada (sinusoidal, triangular, dent de serra, etc.).

Per tant, és necessari desar tots aquests valors, ja que per mitjà d'ells posteriorment podrem recuperar l'objecte de manera satisfactòria. El casos d'ús relatius a passivar connexions es

[1] Per *firing* s'entén processar amb l'algorisme del *Processing object* un determinat número de mostres

refereixen a què per recuperar la topologia original de la xarxa s'hauran de desar els *links* entre objectes. Aquests podran ser entre controls de sortida d'un objecte i controls d'entrada d'un altre, o entre ports de sortida i ports d'entrada.

Com veiem, no és necessari desar les dades que continguin els objectes (mostres de so, frames espectrals, etc.) ja que l'objectiu no és guardar exactament les dades d'una sessió, sinó recrear una topologia de xarxa determinada.

Pel que fa a l'altra tipus d'accions, que serviran per reconstruïr una xarxa passivada, haurem de dissenyar un mecanisme que permeti recrear els objectes tenint en compte la configuració guardada. A més, s'hauran de tornar a connectar ports i controls seguint els *links* serialitzats, de tal manera que la topologia de la xarxa sigui exacta a l'anterior. Finalment, potser s'haurà de crear un primer ordre d'execució de la xarxa creada per assegurar un correcte control de fluxe (per evitar errors de càrrega o topologies impossibles), tasca otorgada al gestor de l'ordre d'execució del sistema.

Cal comentar que aquests casos d'ús provenen del mode supervisat de CLAM. A reacTable* teòricament el gestor d'events centrals serà qui carregui una configuració i vagi demanant la creació dels objectes a la xarxa. Per tant, necessitem mètodes per a la construcció de xarxes amb un mecanisme diferent que el de la serialització. A més l'usuari també desitjarà en molts moments poder modificar la xarxa que s'està executant en temps real.

Control de fluxe

Pel que fa a les connexions, hauran de ser dinàmiques, és a dir, tant les que continguin dades com les relatives a controls hauran d'acceptar canvis en temps d'execució.

En relació a aquesta distinció entre ports i controls, hi ha un requeriment concret del grup de desenvolupament de reacTable*: el sistema haurà de ser capaç de connectar-los junts. Per exemple, agafant la sortida de dades d'un port d'àudio d'un objecte, i transformant les mostres per connectar-ho amb un control d'entrada d'un altre.

Com podem observar al diagrama 3.4, podem afegir o treure connexions i modificar els objectes que hi hagi dins la xarxa (bé eliminant-los o afegint-ne de nous).

Així serà possible completar el requeriment general de crear una xarxa amb un mètode que no sigui el de la serialització, a més de poder modificar la topologia en temps real. A més, cada cop que modifiquem la topologia s'haurà d'esbrinar si es necessita recalcular l'ordre d'execució. Això haurà de ser avaluat quan es modifiquin els objectes, o les connexions relatives al fluxe de dades (els ports), ja que una variació de les connexions de controls no afectarà en cap mesura l'ordre en el qual s'han d'executar els objectes.

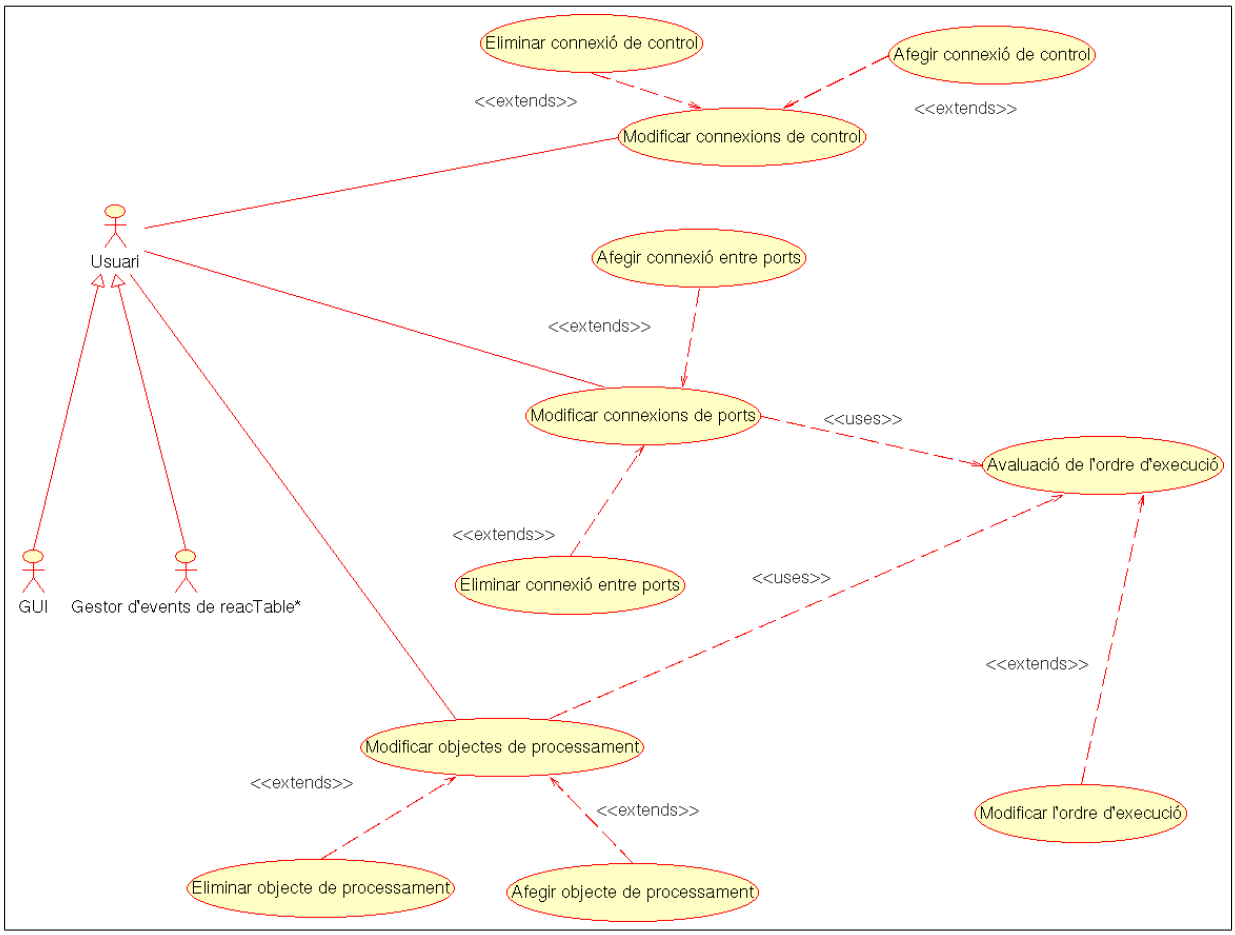

Figura 3.4: Diagrama de casos d'ús de modificacions de la xarxa

Això és conseqüència directa del mètode pel qual es comuniquen els canvis als controls, que és asíncron (al contrari que el fluxe de dades, que és síncron). Aquests conceptes els explicaré de manera més àmplia al capítol relatiu al model del sistema.

Un exemple del que hauria de calcular el gestor que s'ocupi de l'ordre d'execucions el veiem a la figura 3.5

També haurà de ser possible, com hem vist als diferents exemples, crear xarxes amb una barreja d'objectes de processament que treballin amb mostres d'àudio en domini temporal, i d'altres que ho facin amb dades relatives al domini espectral. Això vol dir que serà necessari disposar d'algun sistema que optimitzi les transformacions d'un domini a un altre tenint en compte els objectes que composen la xarxa i els tamanys dels diferents buffers.

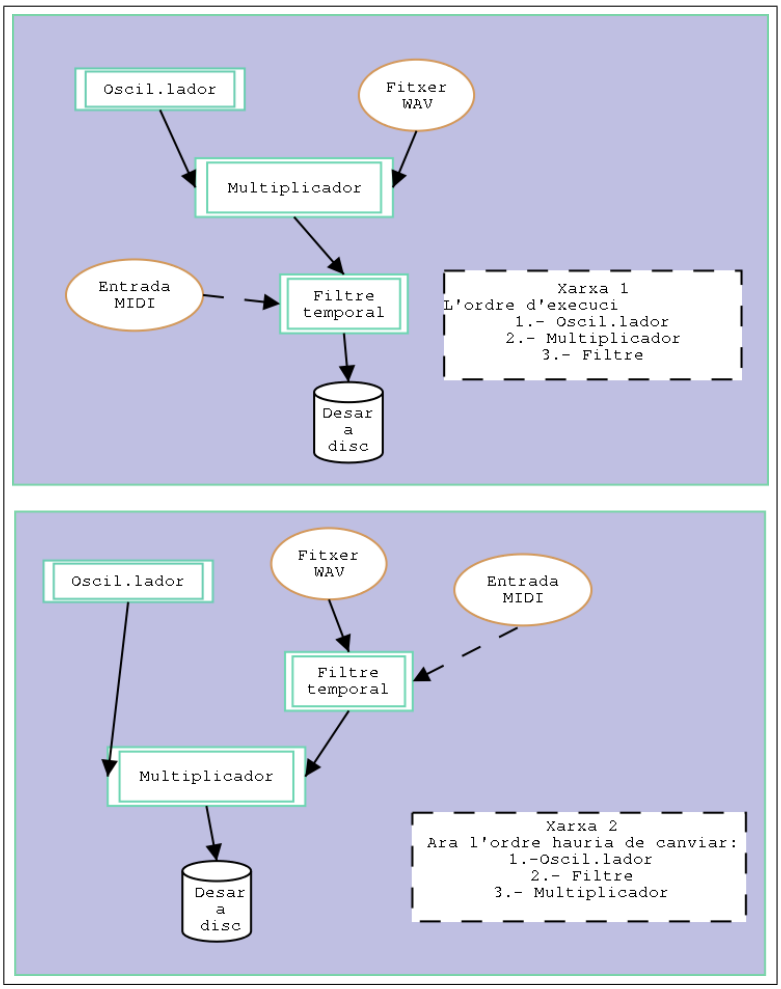

Figura 3.5: Canvi de la xarxa que comporta una alteració de l'ordre d'execució

L'aplicació gràfica

L'aplicació gràfica que es necessita crear pel projecte no és la que farà servir reacTable* (aquesta part està essent desenvolupada per altres membres del grup d'interactius), sinó una aplicació diferent que haurà de córrer en ordinadors normals, sense cap perifèric especial excepte el teclat i el ratolí. Haurà de servir per a crear ràpidament xarxes d'objectes de processament i provar el so resultant, tot permetent experimentar amb els canvis en les connexions de controls i ports.

Per tant, es necessitarà un major control que en la part gràfica final de reacTable*, en què els canvis en la topologia de la xarxa que està sonant són automàtics (donats per unes regles com proximitat, etc.).

No es tracta de recrear a petita escala la taula d'objectes de reacTable*, sinó més aviat de donar un sistema de prototipatge ràpid per a provar sons, combinacions d'algorismes i nous objectes de processament de forma senzilla i intuïtiva.

3.2.2 Requeriments no funcionals

Una de les restriccions del sistema és que haurà de ser en temps real. És a dir, que haurà de ser molt eficient, ja que tota l'aplicació de reacTable* (la part de reconeixement de formes, la part de so, la gestió d'events i la de projecció) hauran de "dialogar" de manera transparent i amb tanta rapidesa com sigui possible. Per tant, el FlowControl sonor de l'aplicació haurà de ser lleuger i ràpid, per tal de maximitzar la relació entre el càlcul de l'ordre d'execució i l'execució en si mateixa. Així, un càlcul estimatori com la divisió T_c/T_{Exec} hauria de donar valors baixos (com més a prop del 0 millor), a on T_c és el temps de càlcul de l'ordre d'execució, i T_{Exec} és el temps d'execució d'un *firing* de tota la xarxa.

A més, els diferents elements del control de fluxe (la xarxa, gestor d'execucions, etc.) hauran de ser bastant independents entre ells. Què vol dir això? Doncs com cadascun dels elements porta a terme una tasca específica, i necessita una informació molt concreta de les altres seccions, els canvis relatius a alguna de les parts no haurien de fer variar gaire les altres. És el que es coneix com a baix acoblament. Aquest fet és especialment rellevant en la relació entre la part de so i la part gràfica, ja que el control de fluxe dissenyat haurà de fer-se servir per una banda en una aplicació gràfica convencional com la que hem apuntat a la introducció, i per l'altra en el sistema de reacTable* complert, amb diverses màquines i diferents sistemes treballant conjuntament.

A més, degut a que no és un sistema desenvolupat tan sols per a aquest projecte, sinó que també haurà de ser útil al framework de CLAM (com a mode supervisat), el sistema necessita ser reusable i, per tant, ha de seguir la filosofia amb què està essent desenvolupada tota la llibreria. Això comporta un alt nivell d'orientació a objectes, compatibilitat entre diferents tipus de dades (buffers de mostres temporals, espectres, controls, estructures superiors com frames i segments, etc.).

Finalment, i degut a la serialització que comentàvem abans, s'haurà de disposar d'algun tipus de sistema que construeixi una xarxa a partir de la descripció en XML que s'hagi de carregar.

Requeriments organitzacionals

Una part molt important dels requeriments no funcionals són aquells relatius a restriccions degudes a l'entorn en el qual es desenvolupa el projecte. En aquest cas, el fet que sigui

dins el Music Technology Group del Institut de l'AudioVisual fa que l´us del framework de desenvolupament CLAM sigui obligatori, ja que és la eina triada per a crear aplicacions d'àudio i música. Relacionat amb això, implementaré la part sonora de l'aplicació en C++ (el llenguatge natural de CLAM) i per tant seguint el paradigma de la orientació a objectes. Finalment, també cal dir que el sistema operatiu sota el qual es desenvoluparà reacTable* serà principalment GNU/Linux. Totes aquestes eleccions del MTG en quant a eines són vàlides per al projecte, com es comenta al capítol d'Eines.

3.2.3 Requeriments del domini

Pel que fa als requeriments del domini, són relatius a conceptes propis del món de l'àudio i la música.

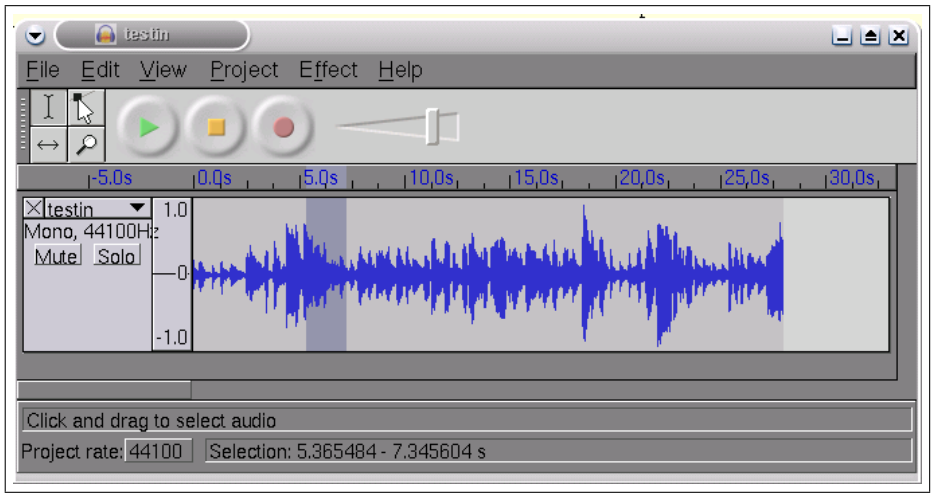

Figura 3.6: Example de representació temporal del so

Bàsicament ens trobem amb la obligació de treballar en un doble domini. És a dir, que en la mateixa xarxa, com hem comentat abans, hauran de conviure d'una banda objectes que treballin amb mostres típiques de so 3.6, i per l'altra objectes que siguin propis del domini espectral 3.7.

Això comporta problemes en quant a què la mida dels buffers que hi ha entre objectes no serà fixa. Tot i que un dels requeriments de reacTable* és que la mida amb la què treballaran els objectes de domini temporal serà donada al inici i no variarà, això ja canvia quan parlem d'objectes espectrals. Aquesta deducció la traiem del fet que, degut al requeriment d'eficiència, la mida dels buffers temporals haurà de ser molt petita en la mesura que sigui possible.

Això afecta directament el temps de latència. És a dir, la distància de temps que hi ha entre

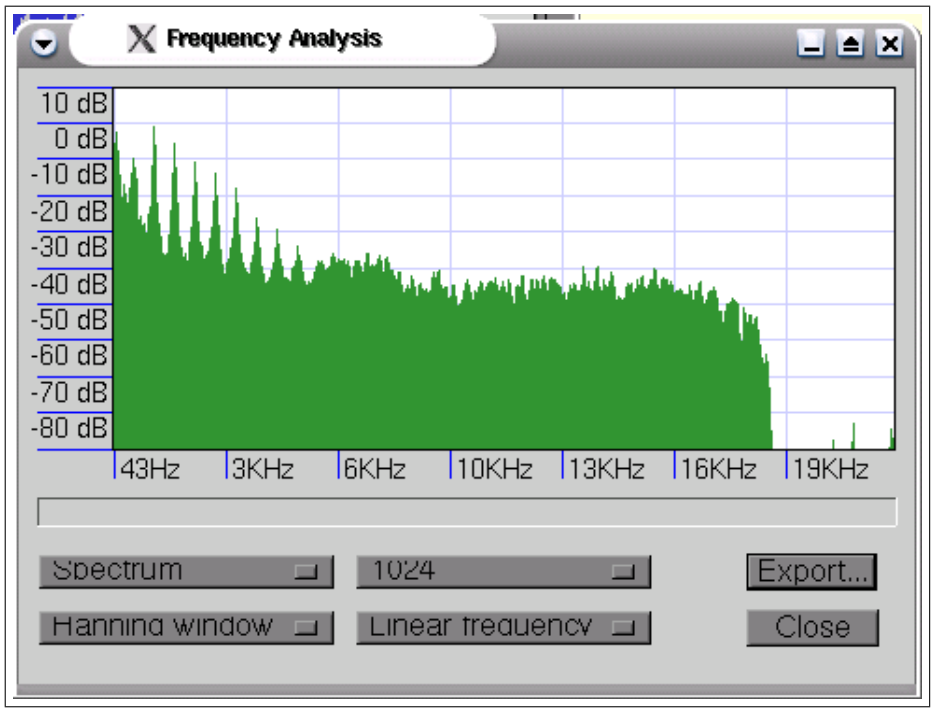

Figura 3.7: Example de representació espectral del so

un canvi al sistema (per exemple fet per una persona tocant un control) i el moment en què el sistema reacciona. Contra més petit sigui el buffer, menor serà la latència, i per tant millor la resposta del sistema. Per contra, buffers petits signifiquen que es necessitaran més recursos de l'ordinador haurà, ja que es requereix una alta potència de processador per tal que la rapidesa al tractar les mostres sigui menor que el temps que dura un búffer *sonant*.

Com a exemple, per a un buffer de 64 mostres en un *sampling rate* de 44100 mostres per segon (aquesta és la mesura típica), tindrem com a màxim 14 milèsimes de segon per a processar-lo.

En canvi, per al processament espectral necessitarem una mida de búffer major, ja que es necessita una "finestra" de com a mínim 256 mostres per tal que el processament tingui sentit. Amb tamanys menors la informació que puguem treure serà molt reduïda, o fins i tot errada.

Com podem veure, degut als requeriments del domini tenim un conflicte de requeriments (entre eficiència i mesura adequada dels búffers de dades) que haurà de ser resolt per un adient disseny de la xarxa i el gestor d'execucions.

Un altre requeriment que necessitarem tenir en compte alhora d'establir el FlowControl és

que serà necessari algun tipus de mecanisme per emmagatzemar mostres antigues, ja que hi haurà objectes que faran servir efectes de *delay*, és a dir, tornar a treballar amb mostres que ja han sonat per aconseguir efectes semblants a ecos, reverberacions, etc.

Finalment hi ha un altre tret conflictiu, aquest cop tan sols particular del mode supervisat de CLAM (reacTable* no ho necessita), que és el tractament de *loops* de so. Això vol dir que el FlowControl haurà de ser capaç de detectar seccions de la xarxa que entren en cicle tancat, i tractar-los de manera especial per tal de no entrar en processos infinits o errades.

Capítol 4

Eines Requerides

Abans de començar a seguir els passos corresponents a l'anàlisi i disseny del sistema que respongui adequadament als requeriments apuntats per les necessitats del projecte, cal determinar quines eines es faran servir , i conèixer-les adequadament per tal de no tenir posteriorment problemes relacionats amb una mala elecció dels recursos emprats, o malgastar esforços en elements no propis del projecte, com podria ser el poc coneixement d'alguna eina important, comportaments no desitjats, etc.

Per tant, he fet una tria bastant exhaustiva dels elements que em fan falta pel projecte, abarcant des d'eines tan bàsiques com el paradigma i el llenguatge de programació que faré servir, fins a eines de desenvolupament com IDE's (*Integrated Development Environment*), editors de textos o els tipus de diagrames destinats a facilitar el disseny.

4.1 Paradigma de programació

Un paradigma, en el sentit històric de la paraula, és *"un conjunt de teories, estàndards, i mètodes que junts representen una forma d'organitzar el coneixement, és a dir, una forma de veure el món"* [7]. Dins l'enginyeria de software, un paradigma de programació és la forma amb la qual el dissenyador d'un sistema implementa una solució al problema que se li presenta. En aquest sentit, el paradigma triat modificarà la forma d'abordar els requeriments, el disseny, i finalment la implementació.

De paradigmes n'hi ha bàsicament quatre:

1. Procedural

   Exemples d'aquest paradigma són C o Pascal. Es basen en una estructura semblant a la que fan servir les CPU actuals en quant a operacions i memòria. És a dir, són llenguatges d'alt nivell que representen una màquina d'estats, i funcionen amb sentències que s'executen de forma lineal i imperativa.

2. Orientació a objectes

   Es basa en una representació del món aplicada a la programació. És a dir, intentar representar els objectes que interactuen al sistema que volem crear, el tipus al que pertanyen, i els missatges que es poden passar entre ells. Així, llenguatges que responen a aquest paradigma com C++, SmallTalk o Java intenten fer un tipus de programació que es dedica a identificar els objectes dels quals està compost el nostre problema, i quina haurà de ser la seqüència de missatges entre ells que li donin solució.

3. Funcional

   La base d'aquest model és purament matemàtica. Com en aquesta ciència, es busca una funció (composta per d'altres funcions) que resolgui el problema plantejat, i que tot aplicant-la al estat inicial que se'ns proposa, farà que trobem la resposta. Alguns exemples són ML o Lisp.

4. Lògic

   Està basat en la lògica de predicats. Enlloc de programar com se soluciona el problema, el que es fa és definir-lo, i els llenguatges com Prolog reproduïran una màquina d'estats no finita amb la qual troben la resposta al problema donat per la definició.

Malgrat que tots els problemes resolubles mitjançant un paradigma de programació poden ser resolts també amb els altres, hi ha elements que fan que una solució amb un model sigui millor que una feta amb un altre, bàsicament perquè les idees proporcionades pel paradigma afectaran profundament el disseny del sistema, i això farà que, variant el model de programació, es puguin simplificar o complicar molt les coses. Per exemple, si pensem

en problemes que comportin cercar la resposta en un gran espai de mostres no del tot definit, com podrien ser aplicacions de tractament del llenguatge natural, serà molt més fàcil treballar amb Prolog, que ja permet aquests mecanismes de cerca, que no pas fer-ho amb C++ o Lisp.

4.1.1 Trets diferencials entre paradigmes

Com veiem, un fet que distingeix els diferents paradigmes és el grau d'abstracció aconseguit. Els llenguatges procedurals permeten generalment menys abstracció, en el sentit que la sintaxi i la semàntica associada a ells no permeten un senzill tractament de les dades de manera associada a un problema. Per aclarir-ho, podem pensar en una operació matemàtica complicada que necessitem resoldre de manera automàtica. Si ho féssim amb un llenguatge estructural, hauríem de crear diferents mètodes per cada membre de la funció, i adaptar el llenguatge al problema. En canvi, si desenvolupéssim seguint un paradigma funcional, l'estructura mateixa del llenguatge abstreu el programador de l'entorn en el qual treballa, permetent-li treballar a nivell del problema més que no pas a nivell de la màquina.

Un altre fet relacionat amb la tria de paradigma respon a l'eficiència necessària en el programa. Bàsicament com més a prop estigui un paradigma del model que fa servir l'ordinador internament (és a dir, basat en sentències), més ràpid serà generalment. Per aquest motiu treballar amb C serà en principi més ràpid que fer-ho amb Lisp o Prolog. Així, existeix un dilema entre eficiència i abstracció del problema. Com més permeti un llenguatge abstracció, tradicionalment menys eficient es tornarà degut als canvis que haurà de generar el compilador del codi per transformar les fonts en el llenguatge propi de baix nivell del computador, que en els ordinadors que usarem és de caire procerdural.

La programació orientada a objectes es situa en un punt intermig entre els dos aspectes realment adequat. Suporta l'abstracció en el sentit que teòricament al definir el sistema no ens hem de barallar amb direccions de memòria, variables o procediments propis del llenguatge estructural. A més, al ser una idea propera al paradigma procedural, és eficient fins al grau que fent servir un bon compilador i un codi ben implementat, no hi ha diferències palpables entre la velocitat d'execució dels dos paradigmes.

4.1.2 Elecció

Pel problema que ens ocupa, la orientació a objectes és una bona tria, ja que necessito un grau d'abstracció elevat per tal de treballar amb mostres de so, controls, control de fluxe, i sobretot idees com els objectes de processament, que per la seva pròpia naturalesa s'adiuen en gran manera a aquest paradigma.

Bàsicament, la idea de objectes que es relacionen entre ells, i que pertanyen a un cert tipus (anomenat *classe*), que a la seva vegada pot heretar trets d'un tipus més general (la seva *superclasse*) és molt adient en el camp de la síntesi i anàlisi de so. Pensem en els diferents algorismes (filtres, generadors de so, etc.) com objectes del nostre model, que es passen entre ells les dades mitjançant un pas de missatges definit amb la série de mètodes que cada classe dóna com a propis (és a dir, publica). A més, els mecanismes comentats d'herència de tipus són perfectes, ja que tots els objectes de processament tenen per una banda coses en comú (com per exemple el mecanisme per parlar entre ells), i per l'altra cadascun d'ells es regeix per un comportament que li és propi, com per exemple l'algorisme que encapsula, el tipus de dades que pot rebre, etc. Per tant, és adequat l'ús de la orientació a objectes pel projecte, tenint en compte a més que és un requeriment no funcional donat per l'entorn en el qual s'ha de desenvolupar el projecte.

4.1.3 Concurrència

Per acabar, actualment hi ha experts en llenguatges de programació que consideren la concurrència (execució de diversos processos a la vegada) com un nou paradigma de programació. Personalment considero que no és tant un nou paradigma, com una positiva evolució dels que ja hem comentat. És a dir, quan es necessita fer servir concurrència, es crearà un programa fet amb un dels paradigmes diferents, però amb un llenguatge que suporti concurrència (com Java o Prolog).

A més, en l'*engine* de so d'aquest sistema no és necessària la concurrència, ja que la part sonora s'haurà d'executar teòricament en una màquina uniprocessador, i en aquest sentit la concurrència tan sols inclouria un cert *overhead* degut a les computacions que necessita, i que per tant faria decréixer l'eficiència. Tan sols es farà servir en l'aplicació gràfica, ja que els cicles que necessiten les dues parts són molt diferents (l'element sonor ha de refrescar-se molt més ràpidament que l'element gràfic).

4.2 Llenguatge de programació

Una vegada decidit el paradigma de programació hem de triar un dels llenguatges que s'adeqüen en aquest model per tal d'implementar la solució proposada en el disseny. D'entre les diverses alternatives, les més representatives són C++ [48], Java, i SmallTalk, per posar alguns exemples. Alguns d'ells, com C++, són una evolució directa des del paradigma de programació procedural. La versió inicial de 1980, anomenada pel seu creador Bjarne Stroustrup [49] *"C with Classes"* i originada a partir del primer llenguatge orientat a objectes, Simula, ha anat creixent i afegint conceptes com herència, polimorfisme fins a ser un dels llenguatges referents cap a la dècada dels 90. Poc després es van homogeneitzar les

diferents versions en un stàndard; 1990 sota ANSI (*American National Standards Institute*), i el 1998 ja amb un caire universal sota ISO (*Internation Standards Organization*).

Per tant, trobem que aquest llenguatge en concret conté elements propis de la orientació a objectes més d'altres que també fan possible l'ús del paradigma procedural o, com diu Bjarne Stroustrup, "I think that *paradigm* is an overused word, and I prefer the less pretentious *programming style*" Aquest fet és una de les principals motius quan es tracta de definir els avantatges i inconvenients d'aquest llenguatge. Al contrari que els altres, és possible mesclar els dos tipus de programació, i això fa que, tot i semblar més complicat per la necessitat d'haver de dominar més elements, el converteixen sense cap mena de dubte en un dels llenguatges més versàtils i flexibles que hi ha actualment al mercat.

4.2.1 Avantatges del C++

Per tant, he decidit fer servir aquest llenguatge per diversos motius que passaré a comentar:

- Flexibilitat Com hem dit, la combinació dels dos paradigmes fan C++ molt més flexible que no pas Java o C#, ja que no estàs subjecte al paradigma d'orientació a objectes en els casos que és més convenient abordar un problema des del punt de vista procedural. Per posar un exemple aplicat, un algorisme de tractament de so podria perfectament ser més fàcil i eficient de programar mitjançant el paradigma procedural dins d'una classe. És a dir, que dins un objecte podem perfectament allunyar-nos de l'abstracció tot fent servir mecanismes de baix nivell propis de C (com punters, ús d'operacions a nivell de bit, etc.) que no són possibles dins els altres llenguatges orientats a objectes.

- Eficiència A més, els requeriments d'eficiència propis d'un sistema de so en temps real com el que necessitem fan molt difícil l'us de Java o C#. Aquests fan servir una màquina virtual intermitja que necessàriament s'executarà més lenta (com a mínim actualment) al no treballar directament amb codi compilat en assemblador, i haver-ho de traduïr en temps real mitjançant programari especialitzat.

- Stàndard Aquest motiu és fonamental, i descarta directament l'ús de Java (que no de C#). Java és un llenguatge propietari de Sun Networks, que no té la denominació stàndard. Això vol dir que el futur del llenguatge és com a mínim incert, en el sentit que en qualsevol moment Sun pot trencar la interfície de programació, o convertir-ho de públic i gratuït a ser un llenguatge de pagament. A més, la lluita entre C# i Java pot decantar la balança cap al llenguatge de Microsoft.

  Per tant, sempre serà preferible l'ús d'eines declarades stàndard per les oficines corresponents (ISO i ANSI), i aquest projecte no és una excepció.

- Compilació Donat que el projecte es desenvoluparà sota plataforma PC més GNU/Linux, l'ús de C# hauria de ser molt discutit, ja que aquest llenguatge, tot i ser un stàndard, és propietat de Microsoft. Això vol dir que, tot i poder-se compilar sota GNU/Linux, la qualitat del codi generat pugui ser posada en dubte. Això no vol dir que necessàriament sigui pitjor, però aquest compilador, anomenat Mono, no té la fiabilitat ni robustesa assegurada com g++, el compilador de C++ de GNU, que ja va per la versió 3.3 i ha demostrat al llarg dels anys una superioritat en quant a qualitat de codi i interpretació de l'stàndard C++ per sobre dels seus rivals.

- Diversitat i suport Al tenir ja bastants anys dins el món informàtic, el llenguatge C++ s'ha anat rodejant per una sèrie de llibreries i ajudes molt superiors en nombre a qualsevol altre llenguatge (potser amb l'excepció de C i els llenguatges d'scripting). A més, moltes d'aquestes són lliures, amb el que podran ser usades en el desenvolupament d'aquest projecte. Entre aquestes utilitats trobem llibreries gràfiques (FLTK, QT, etc.), llibreries de tractament de so (FFTW, CLAM,...), IDE's dedicats a C++ (Microsoft Visual C++, KDevelop), i moltes més utilitats.

Per tant, l'ús de C++ (donat com a requeriment no funcional pel projecte, igual que la orientació a objectes) és adequat més que no pas les altres possibles eleccions, com veiem un cop exposats els motius descrits.

4.2.2 Desavantatges

Com a desavantatges principals, tenim la seva complexitat. Això està íntimament relacionat amb la flexibilitat comentada abans. És a dir, contra més possibilitats se li donin a l'usuari, necessàriament més complexitat hi haurà d'haver en l'eina que es faci servir, ja que per permetre l'aplicació en molts àmbits diferents es fa necessari crear més elements, que afegeixen complexitat a l'entorn. Per posar un exemple, tenim l'ús de punters en C++. Al contrari que d'altres llenguatges orientats a objectes, C++ permet baixar el grau d'abstracció i treballar directament amb direccions de memòria de la màquina amb l'ús de punters. Això, que pot semblar una bona mesura degut a l'eficiència de treballar d'aquesta manera enlloc de referències o còpies, també pot portar molts maldecaps degut a les errades de programació que poden fer accedir a direccions de memòria no vàlides.

Un altre desavantatge és la falta d'utilitats incloses dins el llenguatge vers d'altres com Java o Smalltalk, com podrien ser estructures bàsiques standard, o un "garbage collector". Tot i això, el primer cas ja està resolt gràcies a la Standard Template Library [47]. La STL, implementada originalment pel llenguatge ADA (1979), va convertir-se en part de la col.lecció de llibreries stàndard de C++ al 1994. Està basada en *templates*, que permeten un ús molt flexible en quant a tipus en les operacions d'entrada/sortida, serialització i ús d'estructures com llistes o vectors. Pel que fa al garbage collector, al fer ús de punters C++

ho té bastant més complicat que d'altres llenguatges com JAVA, a on tot ús de memòria és abstret i manegat pel compilador. Tot i això, s'ha assegurat que la pròxima versió de C++ contindrà un garbage collector que gestioni l'eliminació de direccions de memòria no usades i ocupades. Alguns dels problemes relatius a l'ús de punters i la seva eliminació es poden veure a [34].

Tot i això, i observant en conjunt les peculiaritats de cada llenguatge, pel projecte que desenvoluparé el C++ és ideal, degut a la seva eficiència, capacitat d'abstracció i flexibilitat d'ús.

4.2.3 Llenguatges interpretats

Com a apunt final, podríem comentar que es descarta l'ús de llenguatges d'*scripting* tals com Python o Perl per la seva lentitud d'execució i falta de capacitats vers un llenguatge compilat, referents a estructures complexes, comprovació de tipus abans de l'execució i robustesa.

4.3 El so

En aquest projecte hi haurà una part molt important de processament de senyal sonor.

Per tant, de les dues opcions possibles, començar des de zero un motor sonor, o per contra fer servir una llibreria o framework de so ja establerta, s'ha optat per la segona opció. No és raonable fer l'esforç per crear una motor de síntesi i processament sonor si ja existeixen alternatives completament vàlides i que s'ajustin a les necessitats donades. Aquest és el cas que ens ocupa.

4.3.1 Requeriments per la llibreria de so

Els requeriments per la llibreria o framework de so que necessitem són els següents:

- Adaptació a les decisions de paradigma i llenguatge de programació Tot i que no necessàriament obligatori, és clarament aconsellable fer servir una llibreria que s'adapti a l'estil de programació general del projecte. És a dir, existeixen ports per fer servir, per exemple, llibreries de Java dins entorns C++, o també d'altres escrits en C sense gaire complicació, però això pot afectar el disseny en gran mesura, interferint en la manera de plantejar la solució als problemes que se'ns vagin presentant, i creant-ne de nous. Per tant, com a primera mesura s'optarà per llibreries basades en C++.

- Eficiència Tret fonamental en totes les llibreries de so, necessitem que l'eficiència sigui adequada per tal de crear una aplicació capaç de funcionar en temps real, sense retards ocasionats per mala comunicació amb la llibreria.

- Completitud Com que l'aplicació final contindrà elements de molt diverses característiques, tals com tractament de vídeo, projeccions, sortida de so, etc. és necessària més que no pas una llibreria un *framework* que com a mínim doni suport a la bona integració dels diferents elements.

- Lliure Degut a la política del Music Technology Group, el projecte final de carrera serà publicat amb la llicència GPL (*General Public License*). Per tant, la llibreria utilitzada hauria de tenir preferentment aquest mateix tipus de caràcter per evitar complicacions.

Per totes aquestes raons, hi ha diverses eines que es podrien fer servir. Una secció d'elles són relatives a altres programes als quals es podria modificar per tal d'acomplir els requeriments que es necessiten. És a dir, programes com jMax [24] o PD [37], publicats amb la llicència GPL, podrien ser estudiats i el seu codi modificat per tal d'acomodar-se al projecte. Però la pega que hi ha amb aquests programes és, per una banda, que no estan escrits en C++. jMax està desenvolupat amb Java, i PD amb un tipus de C especial anomenat *Objective C* que podríem considerar C amb algunes peculiaritats pròpies de la orientació a objectes. A més, al no ser llibreries, sino aplicacions, la documentació accessible i la facilitat d'ús pot ser raonablement posada en qüestió.

Una altra possibilitat seria Siren [10], però pot ser descartat també per divergència en el llenguatge emprat (Siren és un framework dedicat al processament sonor amb SmallTalk).

Així doncs, cal fer servir un framework de desenvolupament d'aplicacions relacionades amb l'àudio que sigui complet, eficient, basat en el paradigma de la orientació a objectes i a més ha d'estar assegurat que no hi haurà problemes relatius a la llibreria durant el projecte.

OSW (*Open Sound World*) [8] seria una possibilitat, ja que compleix tots aquests requeriments. El problema és que desconec el funcionament d'aquest framework. Això no seria cap impediment si no hi haguessin alternatives, però ja existeix un altre framework amb les mateixes capacitats, i que a més és conegut per mi.

Aquesta eina està essent desenvolupada, com s'ha dit a la introducció, per l'MTG, i rep el nom de CLAM (*C++ Library for Audio and Music*). Tot i que encara està a la versió 0.4, ja és un framework bastant usable i complet com per a desenvolupar aplicacions amb ell. Compleix tots els requeriments del sistema, i a més hi està fortament relacionat, de tal manera que actualment està molt lligat el desenvolupament del mètode supervisat de CLAM (és a dir, CLAM funcionant com a aplicació) amb el disseny del control de fluxe reacTable*.

Per tant, l'ús de CLAM, que és un requeriment organitzacional, és correcte dins aquest projecte per les seves característiques.

4.4 L'entorn gràfic

Com que una part del projecte final de carrera estarà dedicat a crear una interfície gràfica apropiada per a poder editar xarxes de processament, hauré de triar alguna llibreria per tal de no haver de barallar-me amb les crides al sistema necessàries, que complicaria en gran manera la tasca.

A més, en la mesura que sigui possible hauria de ser una aplicació multiplataforma, per tal de funcionar en més d'un entorn, tal i com fa CLAM (actualment en GNU/Linux, Windows i MacOs X). No cal dir que també es requereix de la llibreria a utilitzar que doni la possilibitat de crear productes amb llicència GPL [17], ja que sinó no podria estar fer-se servir dins CLAM.

Per tant, podem descartar com a *toolkit* gràfic escollit les MFC, que tan sols funcionen en Windows i no són lliures, i qualsevol altre llibreria tancada o tan sols suportada per un dels diversos sistemes.

D'entre les que queden, s'hauria de donar preferència a les que estiguin dissenyades en el llenguatge de programació que he triat, és a dir C++ (o en el seu defecte, C). D'entre aquestes n'hi ha dues que sobresurten: Fltk [31] i Qt [50].

Fltk és una llibreria completament lliure dedicada al suport en la creació d'interfícies gràfiques amb *widgets* com les que ja estem acostumats a veure. Els seus avantatges són la llicència, que permet fer qualsevol tipus d'aplicacions com a programari lliure, i sobretot la seva senzillesa. Aquesta senzillesa però, en determinats casos també pot ser un problema per la falta d'estructures més elaborades, que a vegades són necessàries.

Qt, en canvi, és una llibreria concebuda més aviat com un framework, en què se li donen a l'usuari tota una série d'utilitats que van més enllà de la creació de widgets (ús de bases de dades, connexió per xarxa, estructures bàsiques, etc.).

Per tant, és un component més sofisticat que, al contrari que Fltk, pot donar problemes en quant a complexitat i número d'elements a entendre. Tot i això, l'extraordinària documentació de Qt fa que l'ús de la llibreria sigui més assimilable.

Un problema de Qt és la seva doble llicència, sota la qual es poden crear programes sota GPL que corrin en GNU/Linux, però per contra la última versió de la llibreria en Windows té una llicència propietària típica.

4.4.1 Elecció del toolkit gràfic

Després de descartar les altres llibreries semblants, he avaluat quin d'aquests dos candidats és el millor per la meva elecció. M'he decidit per la llibreria QT per diversos motius.

El primer d'ells és el suport de Qt en quant a detecció d'events. Mentre Fltk fa servir mecanismes de *callbacks*, que no són de tipatge segur i poden portar a errors greus i difícils de detectar, Qt implementa una tècnica basada en *signals* i *slots* que, tot i ser semblant a l'anterior, fa un tipatge dels paràmetres a passar a prova d'errors.

Per a entendre el segon motiu hem de pensar en el tipus d'aplicació a fer, i els elements que hem de mostrar. No seran tan solswidgets típics (botons, menús, etc.), sinó que a més tindrem un alt component de dibuixat de formes 2D, tant per a pintar els objectes de processament com les connexions entre ells. En aquest aspecte Qt és superior a Fltk, ja que és una llibreria amb un suport més ampli, en el qual també s'inclouen ajudes per al pintat en dues dimensions.

Un altre element molt important que ha decantat la meva decisió cap a la llibreria de Trolltech és l'excel.lent documentació del projecte. Mentre que la de Fltk és realment petita i bastant deficient, tenint alguns aspectes que tan sols estan documentats a les capçaleres del codi, en Qt hi ha, a més de la documentació classe per classe, tota una série d'exemples i articles comentant les diferents peculiaritats i formes de treballar que té la llibreria.

Finalment, però no un motiu menys important, ja havia treballat abans amb les dues llibreries, i per preferències personals havia estudiat més a fons la llibreria Qt. Per tant, fent servir aquesta no hauré de gastar temps en aprendre com funciona un altre toolkit que en definitiva tingui les mateixes funcions que el que ja em conec.

L'únic desavantatge de fer servir Qt és la llicència doble, però ja que aquest projecte final de carrera es desenvoluparà sota GNU/Linux, així com també el de reacTable*, no és un greu impediment per a fer-la servir.

4.5 Eines de desenvolupament

En aquesta secció hi tenen cabuda totes les eines necessàries per al bon desenvolupament del projecte, tot i que no són pròpiament part d'ell.

4.5.1 Compilador

El compilador que es farà servir és el g++/gcc (GNU C & C++ Compiler), ja que és dels compiladors més estables i que accepten millor l'estàndard. Té tota una comunitat dedicada al seu manteniment i desenvolupament, així que també està raonablement exempte de *bugs*. Es farà servir en la versió més estable existent (2.95) i en la última (3.2), que pot acceptar alguns tipus d'implementacions amb templates que, tot i complir amb l'stàndard, no són acceptades per versions més velles.

4.5.2 Editor de codi

L'editor decidit és GNU Emacs, que com va dir el seu creador Richard Stallman *is the true editor*. Aquest comentari vol dir que no hi ha actualment cap editor, en cap plataforma, que pugui ser adaptat a un ús més ampli del que Emacs té, amb un sistema d'ampliacions basat en Lisp molt robust i adaptable i mitjançant el qual és possible escriure la memòria, el codi, compilar, debugar i fins i tot navegar per l'arbre de directoris sense sortir del mateix programa.

Per tant, fent servir aquest editor de text tan potent no fa falta fer servir diferents aplicacions (una per editar text normal, una per compilar, etc.). Això el converteix en un IDE que, tot i que no adaptar-se al terme convencional referit a entorns com el Microsoft Visual C++ o el KDevelop, compleix tots els propòsits d'aquests i encara bastants més.

4.5.3 Sistema Operatiu

Tot i que el projecte estarà basat en GNU/Linux per les preferències de l'equip de desenvolupament i pels requeriments de baixa latència i distribució del sistema en diverses màquines (una pel tractament de so, una per l'anàlisi gràfic i el reconeixement de formes, ...), hi hauria la possibilitat, donat el caràcter multiplataforma de totes les eines utilitzades, de fer el desenvolupament del projecte final de carrera en Microsoft Windows. Tot i això, per preferències personals de l'autor es farà servir GNU/Linux, ja que és la plataforma òptima per al desenvolupament de les diferents parts de les que està compost el projecte, i no s'estarà subjecte a errors produïts pel propi sistema operatiu que podrien retardar la planificació. A més, les eines triades disponibles per a GNU/Linux són totalment multiplataforma i lliures (amb l'avantatge que això comporta de portabilitat), mentre que les eines *estàndard* del sistema operatiu de Microsoft (Microsoft Visual C++, Visual UML, Microsoft Word) són tan sols per a MS Windows (a més de la despesa econòmica que suposaria poder treballar a casa amb elles si no es pretén piratejar el software). Finalment, un motiu important per fer servir aquest sistema operatiu és l'experiència adquirida en el desenvolupament de software

sota GNU Linux, que fa preferent l'ús d'aquest vers començar a treballar amb Microsoft Windows desde zero en termes de desenvolupament de software.

4.5.4 Gràfics i diagrames

Per expressar amb més claritat les idees de requeriments, anàlisi i disseny que aniran sorgint al llarg d'aquest projecte es crearan diagrames UML (Unified Modelling Language) [44] de diferents tipus (casos d'ús, seqüència, classes,...). Aquest llenguatge és actualment la eina més utilitzada per a diagrames relacionats amb l'enginyeria de programari, per la seva claritat i adaptabilitat a tot tipus de descripcions necessàries pel desenvolupador.

Pel que fa a l'editor de diagrames UML, tot i que encara està per decidir teòricament es farà servir el que, a opinió de l'autor, és el millor editor d'aquest tipus de gràfics disponible en GNU/Linux. S'anomena Umbrello, i tot i que encara està per una versió beta és realment intuïtiu i fàcil d'utilitzar.

Per altres tipus de diagrames s'utilitzarà el programa DIA (també multiplataforma), que permet crear diagrames d'una manera ràpida i general. Aquest també serveix per a crear diagrames UML, però no és l'eina adequada per fer-ho servir de manera intensiva, ja que té un caràcter més general que per exemple Umbrello.

4.5.5 Sistema de control de versions

Per al bon desenvolupament d'un projecte informàtic seriós es fa necessària una eina que permeti la compenetració adient entre tots els seus membres. És a dir, es necessita algun tipus de control de les versions de codi modificades amb dos motius principals. El primer és assegurar que s'està treballant amb la versió actualitzada de les fonts (podent ésser modificada pels altres programadors). L'altre motiu és la necessitat de combinar les modificacions fetes per dos membres diferents sense gaire retards o problemes per l'equip. Aquests dos requisits no serien possibles amb un servei com el donat pel protocol FTP (File Transfer Protocol), ja que no garanteix un servei de *merge* del codi propi amb el que hi hagi al repositori general que fa servir tothom per actualitzar el codi. Per tant, si es fes servir aquest sistema seria impossible que diverses persones treballessin en una mateixa secció en un moment donat. El sistema de versions concurrents(anomenat CVS) sí que ho permet, fent servir mecanismes de *merging*. Hi ha un repositori central a on està la última versió del codi, i cada membre de l'equip que fa servir el CVS té un repositori propi (anomenat *sandbox*) en local a la màquina que fa servir normalment. Es treballa sobre aquest, i s'accedeix al central per tal de sincronitzar la feina feta amb la dels altres membres de l'equip, sigui recollint els canvis fets (amb la ordre *update*), o afegint els canvis propis al codi central (mitjançant la comanda *commit*).

4.5.6 Eines de testeig automàtic

Donat l'estil de desenvolupament que comentaré més tard, i que està basat en part en el testeig del codi a crear, es fa necessària alguna eina per tal de crear tests del codi de manera ràpida i comprovar que funcionen.

Ara mateix la llibreria lliure que permet fer això en C++ és CppUnit [15]. La seva tasca és permetre crear *suites* de tests automàtics de manera ràpida, de tal forma que en qualsevol moment durant el disseny i desenvolupament d'alguna secció del projecte es pugui testejar que el que s'ha fet funciona correctament, i a més incorporar noves funcionalitats plenament testejades al que ja es té.

Com a apunt final, l'ús de CPPUnit està essent introduït dins CLAM com a forma de testeig del codi del framework. Per tant, l'ús de la mateixa llibreria en aquest projecte és adient per tal de mantenir la compatibilitat (si féssim servir una altra llibreria probablement hi hauria alguns problemes).

Capítol 5

El model del sistema

Un cop analitzats els requeriments i objectius del projecte reacTable* i triades les eines que faré servir, arriba l'etapa d'anàlisi del sistema, així com del seu domini. Això vol dir investigar quins són els mecanismes ja implementats per a resoldre els problemes que s'han trobat sistemes semblants al que ens ocupa. D'aquesta manera es poden veure conceptes i solucions proposades a tenir en compte alhora de dissenyar i crear el sistema.

Això donarà pistes sobre dos factors concrets:

- Problemes que ens trobarem durant el disseny.
  Tot i que cada projecte és un món, sí que hi ha una sèrie de punts conflictius amb què es troben tots els projectes amb un domini comú. En aquest cas, hauré d'analitzar programes relatius a la composició musical, i més en concret a aplicacions sonores dissenyades expressament per al seu ús en entorns de temps real.

- Solucions proposades
  Tot i que CLAM i reacTable* tenen els seus propis objectius i requeriments, hi haurà una série de punts o problemes compartits amb altres aplicacions similars.

  Per tant, s'analitzaran quines són les solucions de disseny donades per altres sistemes, especialment pel que fa al control de fluxe i la organització dels elements sonors. També és necessari observar quins són els models gràfics que hi ha en l'actualitat i els paradigmes emprats, ja que és un altre requeriment del projecte crear una interfície gràfica, per tal d'experimentar les possibilitats sonores del motor de síntesi de reacTable*.

Començarem per fer un repàs al programari d'aplicacions musicals i sonores, tot estudiant

els diferents paradigmes existents i els mòduls que caracteritzen un sistema de les característiques de l'*engine* sonor de reacTable*.

5.1 Música dins un ordinador?

Amb la inclusió cada cop major de diferents instruments electrònics dins la música del segle XX, es va anar fent palesa la intenció de fer servir els computadors com a un altre instrument mitjançant el qual crear i processar so.

Un dels motius principals és l'evolució de la música cap a territoris abans desconeguts. És a dir, amb l'ordinador (així com amb altres instruments electrònics més especialitzats) es poden crear sons que serien impossibles d'elaborar amb els instruments d'una orquestra tradicional. A més, l'ordinador conté una enorme capacitat per a emmagatzemar mostres de so i melodies, desar-les i recuperar-les en un format digital de forma segura i sense modificacions no desitjades, i manipular-les de mil formes diferents amb les eines proporcionades a l'efecte.

Amb l'evolució de l'ordinador com a plataforma musical, s'han anat veient diversos avantatges més d'aquesta idea. Entre d'altres, un ordinador permet una varietat d'algorismes de síntesi i processament sonor molt alta, ja que tan sols cal canviar el programari utilitzat per a convertir l'ordinador en un altre tipus d'instrument. És més, fins i tot hi ha la possibilitat de connectar diversos programes per a crear tot un sistema que intercanviï dades en temps real. En GNU/Linux aquest existeix un sistema anomenat Jack que permet dur a terme aquestes connexions de manera senzilla.

Això, combinat amb la facilitat de portar un ordinador (un portàtil per exemple), la possibilitat de connectar-lo amb instruments musicals concrets via el protocol MIDI, i l'augment en les prestacions del computador pròpiament dit, han creat la possibilitat real des de fa una desena d'anys de fer música en temps real mitjançant ordinadors.

En aquest model d'aplicacions és a on podem situar el projecte de reacTable*, i per aquest mateix motiu s'analitzaran els diferents paradigmes per al disseny de programari musical i sonor que han anat creant-se al llarg d'aquesta evolució.

5.1.1 Paradigmes de programari musical

Tot i que inicialment les aplicacions destinades a processament de so en els ordinadors estaven restringides a accions sense temps real per les elevades prestacions que es requereixen, amb l'evolució dels ordinadors cada cop s'han anat desenvolupant més projectes orientat a funcionar en temps real. Així, tindríem dos tipus diferents d'aplicacions.

Per una banda, tenim els sistemes orientats a processament sense temps real.

Dins aquests, hi hauria diversos paradigmes. Per una banda, tenim els programes d'edició de so, destinats més aviat a la producció d'obres musicals que no pas a la composició. És a dir, aquests programes són capaços d'agafar un so prèviament enregistrat i modificar-lo amb un seguit d'algorismes a l'abast de l'usuari, per a produir un so resultant diferent de l'original i adequat als objectius del músic. Un exemple d'aquest tipus d'aplicacions és Ardour [12], un grabador multicanal professional amb llicència GPL 5.1.

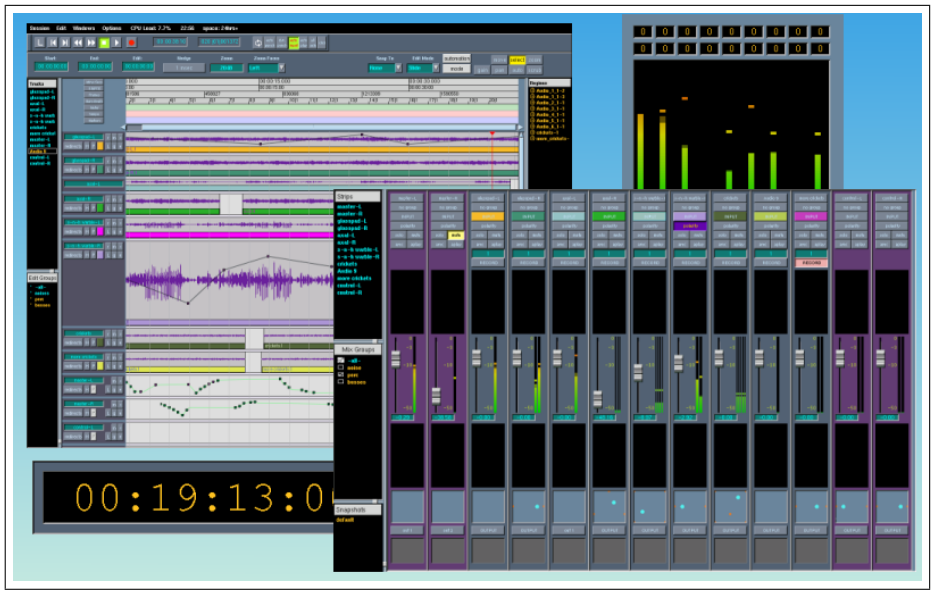

Figura 5.1: Captura de pantalla d'Ardour, extreta de la seva pàgina web

D'altra banda, tenim programes de composició, que volen donar emular la creació de partitures tradicional. És a dir, tot fent servir un llenguatge musical determinat pel sistema, es crearà una "orquestra" d'instruments, detallant la manera en què interactuaran entre ells, i les notes (o conceptes semblants) que hauran de tocar. En aquest paradigma es basa la série de llenguatges N-Music.

Pel que fa als sistemes orientats a temps real, que són els que de fet ens interessa més estudiar, en tenim també de diversos estils. Cadascun d'aquests estils comporta per l'usuari una forma diferent de veure el món de la composició musical.

- Emulació d'una màquina física

  Aquest tipus d'aplicacions no canvien ni varien la forma de tocar un sintetitzador (o qualsevol altre màquina *hardware* musical) real. És a dir, en pantalla es mostra una reproducció d'un instrument d'aquest tipus, i l'usuari interacciona amb ella movent

sliders, prenent botons, etc. amb el ratolí o teclat, tot imitant com es faria amb l'instrument físic.

Figura 5.2: Captura de pantalla de Rebirth, extreta de la seva pàgina web

Aquest tipus d'aplicacions, amb exemples com **Rebirth** 5.2, no té gaire a veure amb el projecte que ens ocupa, ja que de fet no existeixen connexions o control de fluxe dinàmic en temps real.

És a dir, no faria falta establir un ordre d'execució dels diferents algorismes ni esbrinar quin camí segueixen les dades entre instruments perquè ja està predefinit abans de començar a fer-ho servir.

- Mòduls de so connectables

  Aquest tipus d'aplicacions està encapçalada pel sistema de programació visual MAX [39], desenvolupat a partir del 1985 pel grup de Miller Puckette dins l'IRCAM. Basat en un projecte anterior anomenat **RTSKED** (1981, creat per Max Mathews i Pasquale), amb el qual es pretenia crear un sistema d'scheduler en temps real per a un sintetitzador polifònic, MAX va ser l'origen de tota una sèrie de programes basats en el mateix concepte del que s'havia de poder aconseguir amb un programa d'ordinador orientat a la composició musical.

Aquest model es basa en l'encapsulació dels diferents algorismes musicals dins el que podríem anomenar mòduls de so. Aquests mòduls de so es poden connectar (bé mitjançant connexions de dades sonores, o bé mitjançant dades de control) entre ells, a més de poder afegir-ne de nous o borrar els existents.

Podríem dir que aquest paradigma és semblant a la orientació a objectes en enginyeria de software, en el sentit que es basa en encapsular els diferents algorismes de computació en caixes negres, que tan sols són accessibles des de fora mitjançant una interfície "*pública*".

Així, ens trobem amb un model que permet una infinita variació de l'ordre en què estan connectats els diferents objectes, i que en qualsevol moment pot canviar-se alguna d'aquestes connexions, de manera que el so produït variarà.

Per tant, MAX més que un programa de música és una espècie de base sobre la qual desenvolupar les idees que tingui el músic del què hauria de ser un instrument musical. És a dir, es poden crear nous mòduls de so que, combinats amb els ja existents, donin com a resultat una xarxa d'algorismes de processament de so dissenyat per a ser manipulat en temps real pel músic, com un instrument tradicional. La diferència és que amb un sol ordinador es poden crear milers de variacions diferents, sense haver de tenir tantes màquines com configuracions diferents vulgui l'usuari.

Al diagrama 5.3 trobem un exemple de l'apariència que tenen els programes basats en aquest paradigma musical.

Aquest prototipus d'aplicació ha anat evolucionant, i s'han creat diferents programes que, partint de la mateixa base, amplien l'ús al qual pot estar destinat el programa, com PD [38], jMAX [24] o MAX/MSP (del qual abans hem vist una captura de pantalla). Fins i tot s'han creat, desenvolupant aquesta idea, uns sistemes per a la composició musical basats en una estructura anomenada de *plugins*. Aquesta idea consisteix en tenir un programa actuant com a servidor, al qual se li connecten diversos mòduls, cadascun dels quals pot ésser desenvolupat per algú amb uns mínims coneixement de programació, tot seguint unes regles donades per un "protocol" donat, que és el que entendrà el servidor.

Així, donant unes restriccions molt senzilles, es poden crear multitud de *plugins* que es podran connectar i fer servir amb un programa que entengui el protocol utilitzat.

Alguns exemples d'aquesta arquitectura són VST (amb servidors com CUBASE) [46] en Windows i Apple, i LADSPA (Linux Audio Developers Simple Plugin Architecture)[13] en GNU/Linux.

- Nous paradigmes d'interacció musical

  Dins aquesta categoria hi entrarien una série de programes que sense entrar en cap dels altres tipus, intenten crear noves formes de composició sonora, preferiblement en temps real. Com a exemple important dins aquest projecte podríem parlar del programa **FMOL** (Faust Music On-Line) [26], programa desenvolupat pel cap del grup

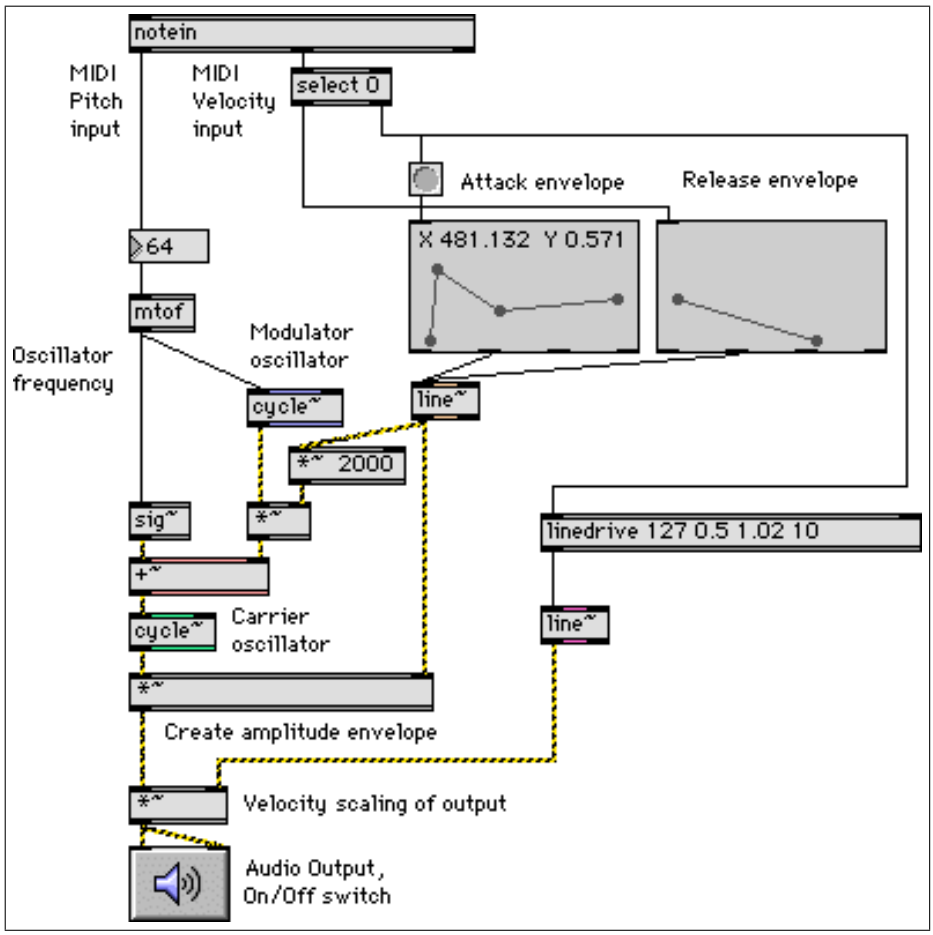

Figura 5.3: Captura de pantalla de MAX/MSP, extreta de la seva pàgina web

d'interactius del MTG, Sergi Jordà. La intenció d'aquesta era crear un instrument sonor que, fugint dels paradigmes clàssics d'interacció home-màquina, pugués aportar noves idees sobre la manera en què es *dialoga* amb un ordinador (en el cas concret d'aquest programa amb tan sols un ratolí) per a crear música d'una forma novedosa, tot obrint nous camins tant en interfícies gràfiques com en la vessant purament sonora [27].

Aquests són bàsicament els paradigmes de programació i llenguatges musicals que existeixen en l'actualitat. MAX és ara mateix, dins la composició en temps real, el paradigma més emprat per a desenvolupar aplicacions de música a nivell sonor dedicades a la composició i improvització en directe.

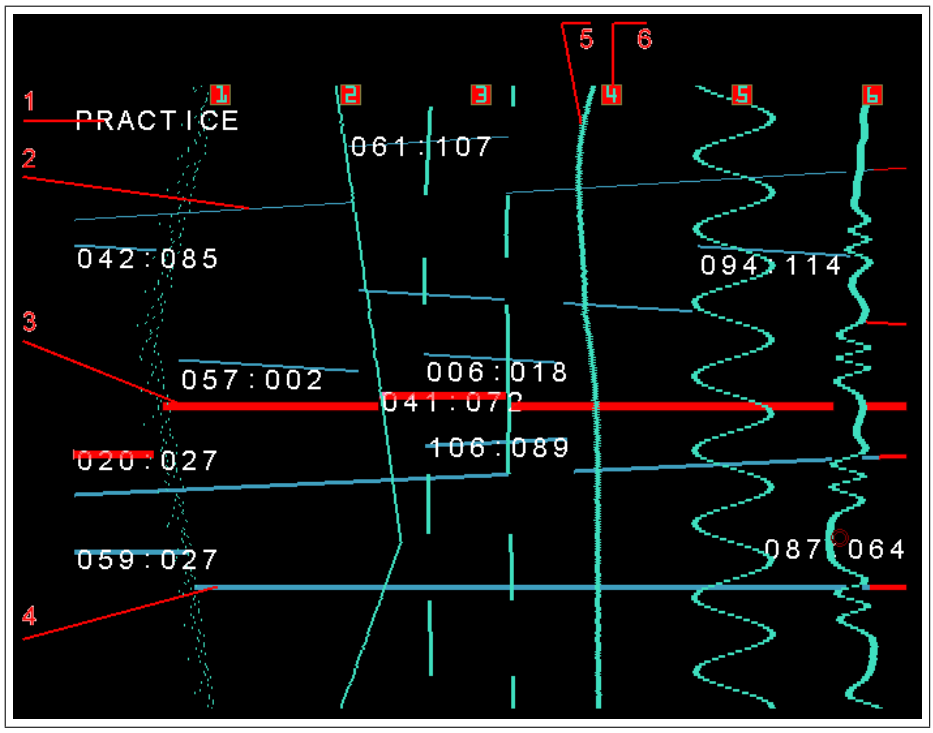

Figura 5.4: Captura de pantalla de FMOL, extreta de la seva pàgina web

5.1.2 La idea dins reacTable*

La idea que ha donat lloc al projecte reacTable* és la de crear un sistema que sigui totalment vàlid tant per a persones sense grans nocions musicals en una instal.lació com per a professionals que puguin fer concerts *exprimint-ne* totes les capacitats musicals.

Així, es pretén crear un model amb un paradigme de caixes (en aquest cas serien objectes sobre una taula) que es vagin connectant en temps real i, així, vagin canviant el so resultant. Si ens centrem en la part d'aquesta projecte final de carrera dins reacTable*, l'objectiu és crear un potent motor de síntesi amb un control de fluxe totalment automàtic que, donades unes regles predefinides de connexió i desconnexió, sigui capaç de registrar l'estat del sistema i les seves connexions, i establir l'ordre d'execució òptim i necessari per al bon funcionament de l'aplicació.

Com veiem, la interacció que proposa reacTable* té punts en comú amb el paradigma dels programes estil MAX, així com també amb les idees que hi ha sota el disseny de FMOL.

És a dir, en principi es pretén crear un control de fluxe semblant al que hi pugui haver en un sistema tipus MAX, amb una xarxa d'algorismes de processament de so connectats

mitjançant links de dades (com mostres de so) i links de controls. La principal diferència radica en què hauran de ser connectats i desconnectats en temps d'execució.

A més, MAX no està dissenyat en principi per a fer servir diferents tipus de dades (tot i que actualment pot processar dades espectrals, etc.). En canvi, el nostre sistema haurà de complir els requeriments relatius a processar mostres relacionades amb el domini temporal i d'altres amb el domini espectral. A més, el tipus de dades que fa servir CLAM poden tenir diversos nivells de complexitat, i en canvi MAX està limitat a tipus de dades molt bàsics.

5.2 El control de fluxe

Centrant-nos en el model d'un sistema basat en mòduls (o caixes negres) com és el que ens ocupa, passaré ara a comentar els aspectes importants en els quals trobaré opcions a considerar en el disseny, a fi que el resultat sigui l'adequat als requeriments que s'han donat.

Començarem comentant els tipus de models de fluxe que es poden construir tenint en compte les especificacions d'un sistema de so en temps real. Bàsicament hi ha dos conceptes bàsics a tenir em compte.

Per una banda tenim l'ús d'un model síncron (amb dades fluïnt amb un temps constant) o asíncron (en aquest cas les dades no segueixen un model de comunicació constant). Per l'altra s'ha d'estudiar com el sistema organitza els dos tipus d'informació que s'enviaran els diferents objectes de processament (controls i dades a processar).

Posteriorment es comentarà la gestió de connexions de dades i controls en CLAM en més profunditat, extenent l'explicació dels mecanismes que ha de fer servir el mode supervisat.

Finalment, es comentaran sistemes relacionats amb la gestió de l'ordre d'execució de diverses tasques, especialment dins aplicacions de música en temps real, així com els tipus de xarxes de processament que s'adiuen al nostre sistema.

5.2.1 Models síncrons i asíncrons

Dins la informàtica hi ha tota una branca d'investigació dedicada a sistemes distribuïts. És a dir, és molt clar generalment com s'hauria de comportar un sistema aïllat dedicat a una determinada tasca i que conté totes les dades i informació que necessita. Però aquest problema d'aconseguir dades necessàries es complica enormement si pensem en diversos elements que interactuant a la vegada, tot formant un sistema complert.

Imaginem un d'aquests sistemes distribuïts compostos d'una série d'elements que necessiten

passar-se informació entre ells, per tal de desenvolupar la seva tasca. Un exemple és la gestió d'un portal, que implica a servidors web, servidors d'aplicacions i bases de dades (com a mínim). Els diferents elements necessitaran informació dels altres, ja que si no de cap manera podran cursar adequadament les peticions dels usuaris. Posem per cas que es demana una web a un dels servidors del portal. Aquest haurà de passar la petició al servidor d'aplicacions, que la trametrà demanant les dades necessàries a la base de dades, i finalment retornarà la resposta al servidor web per tal que l'enviï al navegador del client.

Com veiem és una tasca realment complicada, ja que es necessitarà muntar el sistema de tal manera que es puguin passar dades entre les diferents parts de què està composat el sistema, i a més s'haurà de fer de la manera més ràpida possible. D'altra banda, els diferents elements no es coneixen realment, ja que poden constar d'aplicacions de marques diferents, fins i tot treballar amb formats diferents, etc. Així, serà necessària també una certa independència, per tal que si en un moment determinat es canvia un element de tot el sistema, aquest continuï funcionant. Això és el que s'anomena un sistema desacoblat.

Aquest model de treball és molt similar al que es necessitarà en un problema com el que ens ocupa. Els diferents elements del sistema són objectes de processament, que encapsulen un algorisme amb una configuració determinada, i per tant no coneix a la resta d'elements de la xarxa de la qual forma part. Tot i això, és necessari donar-li algun mecanisme per tal que es pugui comunicar amb la resta d'objectes, tant per a passsar-li's dades (com podria ser per exemple les mostres d'un fitxer d'àudio), com també per a transmetre controls, que varïin algun paràmetre de processament (i que al seu torn es poden passar a un altre objecte, en una espècie de cadena d'events).

Per tant, tenim un sistema basat en elements que necessiten estar desacoblats entre ells, i d'altra banda també requereixen algun mecanisme per a enviar-se dades i controls.

Aquesta informació pot ser enviada de dues formes diferents. Una de les maneres és el que se'n diu un model síncron. En aquest, els objectes esperen que les dades vinguin donades en un fluxe constant en el temps, és a dir, que s'enviïn una certa quantitat de dades en un temps determinat. Per posar un exemple, poden estar dissenyats per a rebre un paquet de 44100 mostres de so cada segon, i per tant cada segon processaran aquesta quantitat de mostres que, a la seva vegada, enviaran als objectes que li demanin dades. Així tenim tot un sistema en el que, fent servir un rellotge comú per a no cometre errades, tots els elements estan "sincronitzats", tot funciona de manera constant i fixa.

Aquest sistema té com a avantatge principal la seva simplicitat, ja que no és necessari implementar cap mecanisme per a esperar nous missatges en qualsevol moment; tan sols rebre cada cert temps ja prefixat una quantitat de dades que ja es coneixen per al seu procés. Per tant, tampoc es rebran diversos missatges de cop.

Per contra, dins els sistemes en temps real aquest model pot imposar retards, ja que tota

la informació necessàriament haurà de ser enviada en les fites que marca el rellotge del sistema com a vàlides. Si, posem per cas, una nova dada s'ha creat i s'ha d'enviar just quan ha passat una d'aquestes fites, s'haurà d'esperar a la següent per tal de mantenir la sincronicitat. A més, també hi pot haver problemes amb el rellotge, especialment si hi ha diverses màquines (cadascuna amb el seu rellotge intern propi), ja que per al correcte funcionament d'un model sincronitzat es necessita que tots els elements tinguin el mateix rellotge.

L'altra manera, ja perfilada comentant el model síncron, és no tenir cap rellotge dins el sistema, i d'aquesta manera els diferents elements enviaran i/o es demanaran dades just quan les necessitin o les hagin d'enviar, sense esperar a cap mecanisme de sincronització. És el que s'anomena un model asíncron.

Com podem veure aquest model evita retards relatius a la necessitat d'enviar les dades en moments determinats pel sistema, ja que es poden enviar sempre que calgui. De fet no necessitarem ni un rellotge.

Per contra, pot arribar a ser considerablement més complicat que el model síncron, ja que es necessiten mecanismes per a saber rebre i enviar les dades quan es necessiti, i per tant haurem de saber quan ens estan enviant dades. Una manera de saber-ho és anar preguntant de manera constant si hi han missatges nous per a nosaltres. És el que s'anomena un mecanisme de *polling*. Tot i que sembla raonablement senzill, comporta una sobrecàrrega del sistema, ja que es va preguntant constantment si hi ha missatges nous. Si per exemple, tan sols rebem dades noves una de cada 1000 consultes, es veu clarament que estem desperdiciant bona part del temps de CPU.

manera de fer-ho és implementant un mecanisme mitjançant el qual hi hagi un avís quan haguem un missatge, sense haver-ho d'anar preguntant cada cop. Hi ha moltes maneres de crear un model així, com podrien ser les interrupcions, les *callbacks* o els models de signal/slot. Les interrupcions es basen en què el receptor del missatge atendrà a una série de peticions ja creades anteriorment (les senyals o interrupcions). Quan salta una d'aquestes interrupcions, s'executa un codi en concret.

Les callbacks i el model signal/slot són ampliament utilitzats en sistemes asíncrons amb interacció amb l'usuari, com poden ser les interfícies gràfiques. Consten d'una funció o *slot* a la qual hi ha una série d'accions (igual que la que podria executar-se en una interrupció). A aquest mètode o funció se li poden passar arguments, que serien les dades vingudes de, per exemple, un altre objecte de processament. Així doncs, tenim una manera d'enviar dades asíncronament i sense carregar el sistema. Són els mecanismes orientats a events.

La diferència entre els dos tipus, que són bastant semblants, és que amb una implementació de signal/slot els paràmetres que passem ja estan predefinits per la senyal que s'enviarà per a cridar l'slot. Per contra, a la callback els tipus de paràmetre no són predefinits amb un

tipatge estricte i, per tant, no són segurs (es podria passar algun element que doni un error, sense que es pugui evitar en temps de compilació).

5.2.2 Fluxe de control vs fluxe de dades

Un cop analitzats els tipus de fluxe que es poden donar, és hora de passar a veure com poden afectar a les dades que enviarà el sistema que ens ocupa. Com hem comentat, bàsicament tenim dos tipus de missatges. Per una banda, tenim els missatges de control. Aquests poden provenir d'alguna font externa al sistema (com interfícies gràfiques, instruments MIDI, etc) i, per la seva naturalesa, són eminentment asíncrons. Això vol dir que en qualsevol moment podem esperar nous missatges de control, i com que estem en un ambient en temps real, hauria de ser possible que els elements del sistema que depenguin d'aquest control coneguin el canvi de la manera més ràpida possible. Per tant, un sistema de so en temps real necessitarà lògicament un model de fluxe asíncron. Tot i això, no ha estat fins a aplicacions com MAX que s'ha pogut implementar un disseny així, ja que, com hem comentat abans, un model asíncron és més complicat, i a més necessita més recursos computacionals que un de síncron.

L'altra tipus d'elements a passar són les dades a tractar pròpiament dites. Aquestes poden ser, per exemple, mostres de so provinents d'un fitxer que s'està processant. També podria donar-se el cas, donats els requeriments que hem comentat, que aquestes dades no siguin del mateix tipus. És a dir, que alguns dels objectes de processament esperin com a entrada dades de domini temporal mentre que d'altres objectes necessitin mostres del domini espectral. Això comporta una série de transformacions que poden fer créixer la complexitat del sistema.

D'altra banda, aquest tipus de dades compleixen un comportament síncron. És a dir, donada una xarxa d'objectes de processament, les dades que circulin a través d'ella hauran necessàriament de passar a través del camí donat, i ho faran de manera constant, ja que els sons que escoltem ho són. Per a veure-ho més clar podem pensar en com es reprodueix una cançó en un aparell de *compact discs*. En aquest, mentre que les accions de control que hi puguem dur a terme són completament asíncrones (pujar o baixar el volum, canviar de disc, etc.), la cadència a la que sóna l'aparell un cop pitjat el botó de *play* és constant. La xifra normal és de 44100 mostres per segon. Si no fos així estaríem distorsionant la música grabada. Per tant, un sistema de so en temps real tindrà lògicament un comportament síncron en quant a les mostres que està processant, a fi que el resultat sigui correcte i no s'introdueixin errors.

5.3 Avaluació de l'ordre d'execució

Després d'estudiar els diferents fluxes (dades, controls, síncrons, asíncrons,...) que es podran donar en el nostre sistema, és lògic pensar que, ja que tot això ha de controlar-se de forma automàtica donat els requeriments de reacTable* i el mode supervisat, necessitem algun element que s'ocupi precisament d'això: avaluar l'ordre d'execució dels algorismes existents del sistema.

Així, és necessari establir un ordre pel qual es vagin executant els diferents objectes de processament, sempre tenint en compte els fluxes de dades que hi hagi entre ells. Això, que pot semblar senzill en xarxes petites o sense gaire complicacions (per exemple, la figura 5.5), creix en dificultat quan ens adonem que poden existir topologies amb diversos camins que es van dividint i unint al llarg del procés, a més de contenir *loops* a on no tenim un sentit clar cap al que hagin de *circular* les dades.

Figura 5.5: En aquesta xarxa l'avaluació de l'ordre d'execució seria senzill

Podem percebre totes aquestes dificultats per a establir un ordre d'execució concret en la figura 5.6.

Aquest problema era inexistent fins fa poc, ja que és un problema molt relacionat amb les aplicacions en temps real. Les aplicacions a on no es dona aquesta restricció han de planificar l'ordre en què es van executant els algorismes de forma manual (és a dir, qui soluciona aquest problema és el propi programador o usuari) o potser abans de començar a processar les dades (podríem dir en temps de configuració).

Per contra, els programes en temps real (i especialment els programes dissenyats sota el paradigma de mòduls connectables com MAX o el que ens ocupa) necessiten algun tipus de planificació que respongui als canvis en la topologia de la xarxa, tot recalculant l'ordre d'execució dels objectes. A més, el sistema no pot gastar gaire cicles de CPU en això, ja que no es poden permetre tenir retards en la sortida dins una aplicació de so en temps real. Si passés això l'efecte és semblant a tenir un búffer de mostres massa petit: se senten *clicks* degut a què no estem processant totes les mostres que necessitem.

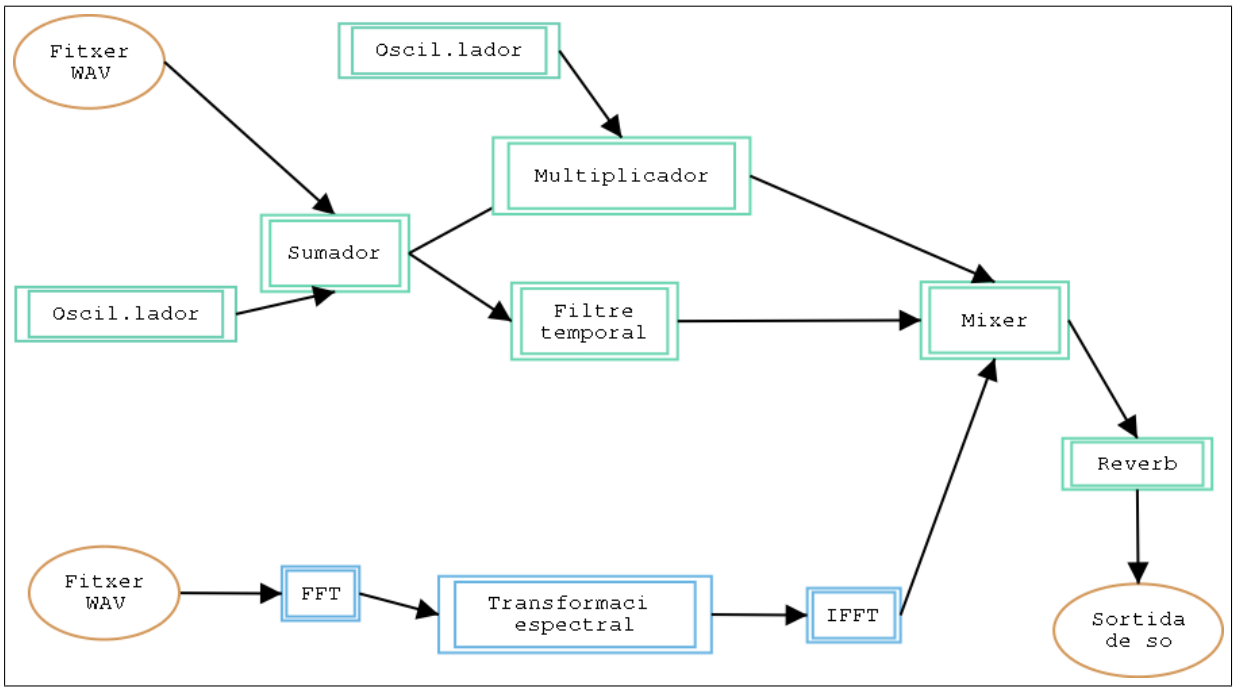

Figura 5.6: Aquí ens trobem en una avaluació bastant més complexa

5.3.1 MAX

Si observem sistemes semblants, trobem que tota la família MAX fa servir bàsicament el mateix algorisme d'*scheduling*. Es determina l'ordre d'execució per la disposició gràfica dels objectes. Així, es recorre la llista d'objectes, executant abans els que estiguin a dalt i a l'esquerra, per anar passant després cap a baix i a la dreta. Aquest sistema tan senzill dóna un resultat molt bo, ja que degut a la seva simplicitat no demana gaire recursos, i a més fa que tots els usuaris de MAX i PD comprenguin com funciona l'aplicació sense necessitat de saber programar.

Per contra, com veiem es delega una part de la complexitat del scheduler a criteri del usuari, i per tant podríem considerar que és un planificador semi-automàtic.

No és adequat pel nostre sistema, ja que es pretén que reacTable* tracti automàticament l'execució del sistema, tot permetent a l'usuari experimentar amb la música sense haver de preocupar-se d'altres problemes.

5.3.2 Schedulers amb models amb prioritats

Alguns tipus d'scheduling, semblants als que es fan servir per a repartir el temps entre processos en un sistema operatiu, donen un ordre d'execució relacionat amb una prioritat donada per cada objecte. És a dir, cada mòdul té una prioritat en funció de la seva importància, i això es té en compte a l'hora de calcular l'ordre, executant-se primer els que tinguin major prioritat i fent després els restants.

Aquest sistema no és gaire eficient per a aplicacions de so en temps real, ja que com hem vist tots els objectes són igual d'importants en una xarxa, en el sentit que si no s'executa el primer en una cadena d'algorismes, difícilment podrem executar els que van després, per molt que siguin més importants.

5.3.3 Sistemes basats en topologies de graf

Hi ha tota una série d'algorismes que enlloc d'intentar establir un ordre d'execució basat en prioritats o posicions gràfiques, fan servir el graf de connexions dirigides entre objectes per tal d'establir quins objectes precedeixen als altres.

Aquests, que podem trobar en branques de la informàtica relacionades amb autòmats finits i fluxes de dades [9], es divideixen bàsicament en dos tipus principals: cerca en profunditat i en amplada.

Això vol dir que, donat un objecte en concret (un node del graf), podríem començar a buscar en un fill en concret (i en un fill d'aquest, recursivament), establint una política de profunditat abans que d'amplada, o pel contrari anar buscant entre tots els fills d'un mateix pare.

D'entre aquestes dues opcions, en el model que ens ocupa és més eficient la cerca en amplada, ja que tornant al problema que comentava en l'apartat de models amb prioritats, si anem intentant executar els objectes en profunditat no podrem avançar si un pare d'algun d'ells no té dades per donar-li.

Per tant, un algorisme d'aquest tipus amb cerca en amplada seria un bon candidat com a algorisme d'establiment de l'ordre d'execució. A la part de disseny comentarem més àmpliament un algorisme proposat seguint aquesta tècnica.

5.4 CLAM: C++ Library for Audio and Music

La base principal del sistema que s'haurà de desenvolupar és el framework creat per l'MTG anomenat CLAM [14]. Aquest projecte va començar essent una llibreria de propòsit intern del Music Technology Group. Es pretenia obtenir una potent eina per a desenvolupar aplicacions relacionades amb la síntesi i el processament de so i música. Els objectius bàsics eren:

1. Crear un framework de desenvolupament complet.

   CLAM havia de ser bastant més que unes quantes classes per a processar so o una llibreria. S'havia de donar suport per a totes les necessitats d'un projecte relacionat amb l'àudio.

   Això vol dir donar suport per a la serialització i passivació de les dades, un sistema de visualització general per a poder estudiar la informació sonora en format gràfic, suports d'entrada i sortida per a fitxer, format MIDI, etc.

   Actualment ja estan implementats la majoria d'aquests mòduls de CLAM. En concret, la serialització de les dades es porta a terme fent ús dels format stàndard XML i SDIF, i el model de visualització permet crear ràpidament programes dissenyats amb els *toolkits* gràfics FLTK i QT.

2. Flexibilitat per a adaptar CLAM a totes les necessitats del MTG.

   Ja que aquest grup de recerca té equips amb objectius i àrees d'investigació molt diversos, el framework general del MTG ha de ser flexible, en la mesura que pugui ser utilitzat i adaptat sense gaire esforç per a tot tipus d'aplicacions relacionades amb el domini del so.

3. Necessitat de crear un producte multiplataforma.

   Seguint el punt anterior, els projectes del MTG són desenvolupats en un ampli ventall d'arquitectures (PC amb sistema operatiu Microsoft Windows, PC funcionant sota GNU/Linux i plataformes desenvolupades per Apple com Mac OS X). Així doncs, el framework ha de donar resposta a totes les plataformes, i el codi desenvolupat sota un compilador en concret ha de poder ser compilat també sota els altres.

   Per aquesta raó es fan servir toolkits gràfics multiplataformes com els comentats anteriorment.

Aquest projecte ha esdevingut més important encara amb la introducció del framework dins el projecte europeu AGNULA [1]. Aquest projecte intenta crear una distribució del

sistema operatiu lliure GNU/Linux totalment composta per programari lliure i orientada a professionals del món de la música i el so.

Així doncs, CLAM ha estat publicada amb llicència GPL (*General Public License*), i ha passat de ser un producte dedicat exclusivament a l'ús dins l'MTG a ser obert amb tots els drets i deures que garanteix aquesta llicència.

Un cop comentats els objectius del framework, passarem ara a explicar quins són els components bàsics de CLAM, i més concretament aquells que estan directament relacionats amb el projecte que ens ocupa.

5.4.1 Dynamic Types

Aquest concepte és un dels pilars sobre el qual es fonamenta CLAM. Tenint en compte els requeriments relacionats amb el camp del so, s'ha creat un mecanisme per a poder aconseguir *tipatge dinàmic* d'atributs dins les classes C++ que formen el nucli de CLAM-

Qué vol dir això? Doncs senzillament que, donat un objecte d'alguna classe que contingui una série de dades, podrem instanciar i desinstanciar els seus atributs en temps d'execució.

Per exemple, si tenim un objecte d'una classe anomenada *Spectrum* (que conté dades relacionades amb l'anàlisi espectral d'un fragment de so), podem expressar la seva informació de moltes maneres diferents (en valors absoluts, coordenades polars, etc.).

Qué passa si per exemple sols volem una d'aquestes representacions? Doncs en el C++ normal hauríem de tenir una classe amb molts atributs, dels quals tan sols en farem servir una petita part. L'altra opció seria anar dividint *Spectrum* en diverses classes, cadascuna amb una representació. Però això faria augmentar la complexitat (ja que es multiplicaria enormement el número de classes, ja de per si prou gran), i també disminuiria la cohesió del disseny, ja que estaríem disgregant informació relacionada que hauria de conformar un sol objecte.

Els *Dynamic Types* donen una adequada solució a aquest problema de flexibilitat *versus* complexitat. Es tracta de crear, mitjançant l'ús de macros semblants a les usades en C i *templates*, un mecanisme que permeti tenir dins un objecte tan sols aquells atributs necessaris en un moment determinat. Així, si sols volem fer servir coordenades polars, tan sols tindrem l'atribut que contingui aquesta informació. Si en un altre moment ens interessa més una altra representació de l'espectre, sempre podrem desinstanciar l'atribut anterior i instanciar la que més ens convingui, o fins i tot mantenir vàries representacions.

Com veiem és una potent eina que permet un alt grau de flexibilitat de les dades en CLAM sense perdre cohesió ni forçar dissenys il.lògics.

5.4.2 Processing Objects

Els anomenats objectes de processament són el nucli de CLAM. Bàsicament, un Processing Object és un algorisme de processament de so encapsulat dins una classe que permet modificar la configuració dels paràmetres de l'algorisme, a més d'assegurar la comunicació amb d'altres objectes de processament mitjançant ports i controls (que seran comentats posteriorment). És lògic pensar en una aplicació de so com en una cadena d'objectes de processament, als quals se'ls passa una série de dades (com un fitxer de so, les mostres grabades d'un micròfon, etc.), i aquests les van processant, passant-se un a l'altre les dades després d'aplicar-li un algorisme.

Això és bàsicament el que possibilita l'estructura de la classe *Processing* (o Processing Object). Proporciona la interfície necessària per a interactuar amb un algorisme, tot encapsulant-lo per tal de crear un disseny intuïtiu i potent. Aquesta classe consta de diversos components. Un d'ells és la configuració, a on es guarden els paràmetres amb els quals funciona l'algorisme (i que podrem modificar). Un altre és el mètode **Do**, és a dir, la funció en la qual es processa una certa quantitat de dades (és a dir, un número concret de instàncies de la classe Processing Data que comentaré pròximament). Finalment podem modificar alguns paràmetres de l'objecte de processament mitjançant controls (com, per exemple, seleccionar un algorisme de síntesi de sinusoides d'entre una col.lecció de possibles tries).

A la figura 5.7 veiem l'estructura i els elements que componen un objecte de processament (diagrama extret de la documentació de CLAM).

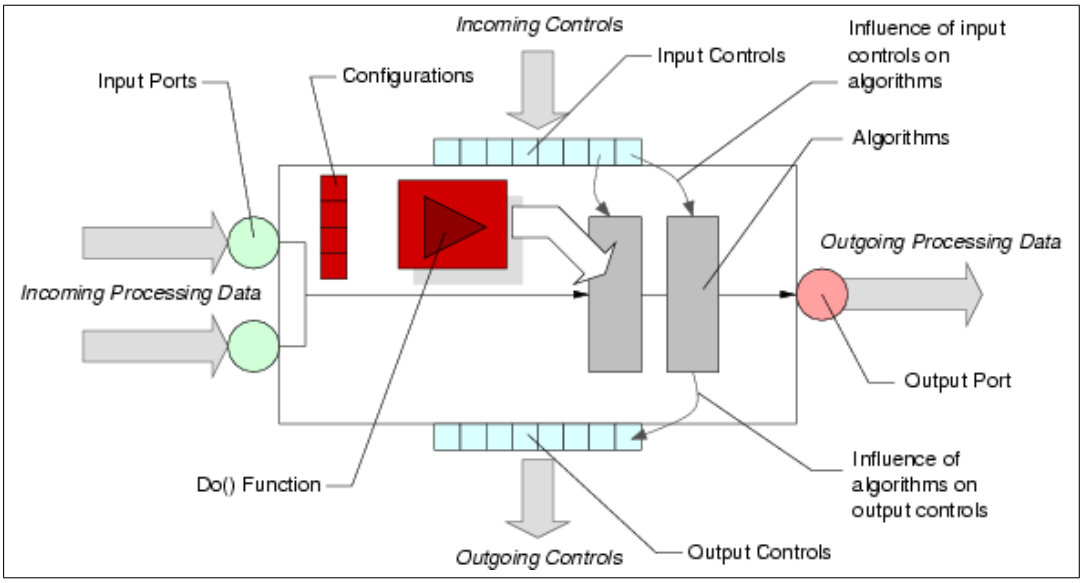

Figura 5.7: Estructura d'un object de processament

5.4.3 Processing Data

L'altre element bàsic de CLAM són els anomenats Processing Data. Donats els requeriments del MTG, grup que treballa profundament en l'ús del domini espectral per a crear aplicacions de so, era necessari tenir més d'un tipus de forma amb la qual representar la informació. Això no és necessari en projectes enfocats en alguna àrea concreta del processament de so, però en el cas del framework CLAM s'han de disposar de recursos per a tenir en una mateixa aplicació molts tipus de dades d'origen radicalment diferent (fitxers MIDI, mostres de so, o fins i tot models ocults de Markov...).

Així doncs, els Processing Data són classes que, tot implementant els mecanismes de *Dynamic Type* comentats abans, permeten encapsular les dades per tal que els diferents objectes de processament les puguin consultar i manipular de forma flexible i senzilla.

5.4.4 Modes de funcionament

CLAM haurà de tenir un doble funcionament. Per una banda, es necessita una llibreria per a poder desenvolupar aplicacions relacionades amb el so i la música. És a dir, una col.lecció de classes i funcionalitats que ens permetin crear un programa sense haver de partir de zero. És l'anomenat mode no supervisat, en el qual el programador haurà de decidir en quin ordre s'executaran els algorismes per a processar l'àudio, i quines relacions tindran entre ells. És el mode que està actualment en funcionament.

L'altre mode és l'anomenat supervisat. En aquest es pretén crear un sistema de prototipatge ràpid, amb el qual algú sense coneixements de programació pugui ser capaç de crear objectes de processament, enllaçar-los com cregui oportú, i en definitiva crear prototipus d'aplicacions de processament de so sense haver de programar en cap cas. Aquest mode és el centre d'aquest projecte, ja que caldrà dissenyar el seu funcionament, així com adaptar-ho a les necessitats del projecte reacTable*.

5.4.5 Ports i controls

Hi ha diversos mitjans pels quals una entitat externa (com pot ser un usuari, una interfície gràfica, ...) pot modificar el comportament i les dades que està emprant un objecte de processament. Bàsicament aquests són els quatre mètodes:

- Configuració

  Com hem dit abans, cada objecte de processament té una configuració associada, que conté tots els paràmetres pels quals es regeix el comportament de l'algorisme

encapsulat.

- Argument del mètode Do

  Quan es fa servir CLAM com una llibreria normal de processament, al mètode que serveix per a executar l'algorisme (anomenat **Do(...)** se li passen com a arguments els *Processing Data* que haurà de modificat, tants els d'entrada, com els de sortida. Per exemple, si tenim un Processing que suma dos fragments d'àudio, al Do li passarem dos objectes Audio amb les mostres d'entrada, i un altre a on emmatgatzemarem la sortida.

- Ports

  El mecanisme del punt anterior ha d'automatitzar-se necessàriament en el mode supervisat de CLAM. Així, s'ha creat una interfície comuna per la qual podem entrar dades per a processar dins un Processing Object. Són els anomenats Ports.

  Aquests són relacionats amb alguna estructura que contingui les dades necessàries d'entrada de l'algorisme (per exemple, ProcessingData o nodes, que comentarem més tard) abans de començar l'execució. D'aquesta manera es podrà executar el mètode **Do()** sense paràmetres (ja que hem especificat abans les entrades i sortides).

- Controls

  Finalment, també podrem afectar un Processing object mitjançant controls. Els controls en CLAM serveixen per a modificar tots aquells paràmetres que són susceptibles de canvi en temps d'execució. Així per exemple, si tenim un sumador com el comentat abans, podrem disposar de controls d'entrada que facilitin modificar els valors de guany dels dos fragments de so que estem mesclant, o fins i tot silenciar completament un dels dos.

5.4.6 Nodes

Hem dit que els Processing Data són les estructures en les quals "empaquetem" les dades que necessiten els Processing Object per a aplicar l'algorisme corresponent. Això, que sembla obvi en el mode no supervisat de CLAM, comença a complicar-se al pensar en el mode supervisat. Qué passa si hi ha diversos objectes de processament llegint d'un mateix port de sortida d'un altre Processing? És necessari duplicar el Processing Data tants cops com objectes vulguin consumir les dades? A més, haurem d'enllaçar tots els objectes al productor?

Com veiem, això pot comportar greus problemes d'eficiència, ja que com hem comentat abans el mode supervisat està dissenyat per a poder-se executar en temps real. I evidentment duplicar informació innecessàriament no és gaire eficient.

Així s'ha dissenyat en CLAM l'anomenat mecanisme de nodes. Es basa en què entre cada objecte de processament i els que consumeixen les dades que ell produeix hi hagi un node que sigui el supervisor de les tasques de lectura i escriptura de dades en els Processing Data. Com veiem, són necessaris mecanismes per a evitar duplicar informació i afavorir l'intercanvi de dades entre objectes de manera transparent. Aquests són els objectius dels nodes: crear un sistema pel qual es maximitzi l'eficiència d'un intercanvi de dades entre objectes de processament sense fer perillar la consistència de les dades.

Aquest últim punt és especialment important. S'han d'evitar problemes relatius a pèrdua de dades útils i colls d'ampolla. Per exemple, podria donar-se el cas que un objecte de processament produeixi dades a una velocitat major de les que les consumeix un altre objecte. Hi hauria un moment en el què s'estarien sobreescrivint dades que encara no s'han consumit. Aquest problema, i d'altres relacionats amb la comunicació d'elements productors-consumidors com els típics d'aplicacions *multithreading*, és a dir, amb diferent fils d'execució, amb els quals aquesta relació entre objectes de processament productors de dades i consumidors, han de ser sol.lucionats pels nodes de CLAM.

5.4.7 Xarxa

Relacionat amb aquest punt sobre els nodes, hi ha el concepte de xarxa, estretament relacionat amb el mode supervisat de CLAM. Dins una aplicació que funcioni sota aquest mode, tindrem bàsicament un graf de connexions entre una série d'objectes de processament, que donen com a sortida una transformació de les dades d'entrada fins a la sortida desitjada.

Per tal que es pugui automatitzar un model amb el qual totes aquestes connexions entre objectes siguin tractades de forma automàtica, necessitem algun tipus de representació d'aquest graf de connexions. És l'anomenada xarxa de processament.

En aquesta trobarem una representació de l'estructura de connexions del graf, formada pels objectes de processament, els nodes que els comuniquen seguint els enllaços marcats per l'usuari, i les connexions de control.

Com podem observar és una estructura molt útil en un cas com reacTable*, en el qual necessitem mètodes per a modificar en temps d'execució una série d'objectes de processament i les seves connexions. Totes aquestes funcionalitats hauran de ser suportades per la xarxa, la qual estarà mantenint en tot moment una representació actualitzada del graf de connexions del mode supervisat.

També serà extremadament funcional una serialització d'aquesta xarxa, en la mesura que així en qualsevol moment es podrà passivar l'estat de connexions, per a poder-lo fer servir posteriorment.

5.4.8 Sistema complert

El sistema complert, tenint en compte els diferents elements que hem anat comentat, seria el que vé donat pel diagrama general del sistema que trobem a 5.8.

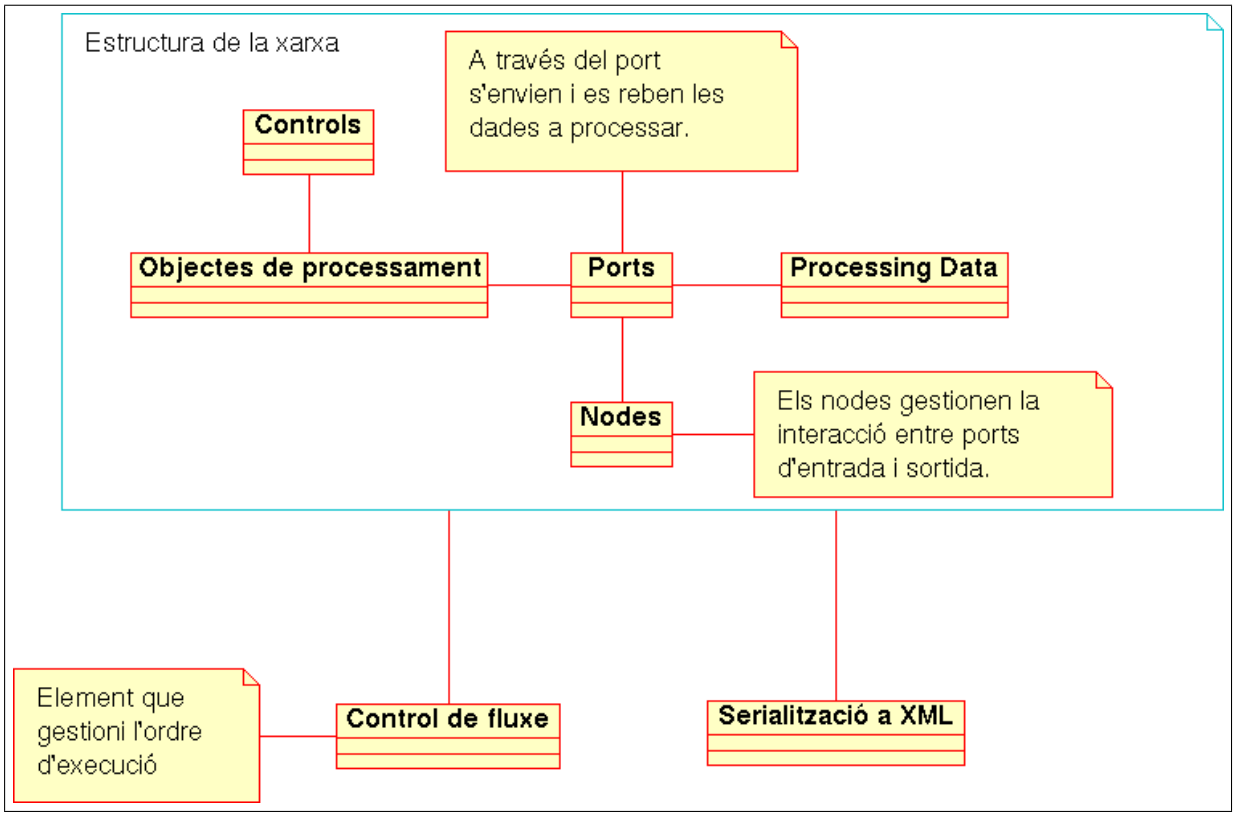

Figura 5.8: Estructura del elements que componen el control de fluxe

Com podem veure, tenim una col.lecció d'objectes de processament, cadascun amb una série de ports i controls d'entrada/sortida. Aquests objectes es comuniquen es passen instàncies de Processing Data entre ells a través dels Ports, mitjançant l'ús de nodes. A la seva vegada, aquests nodes gestionen les dades que li arriben d'escriptura (provinents d'un port de sortida d'un objecte) i de lectura (s'envien a ports d'entrada d'altres Processing).

En un nivell més alt, la xarxa és qui conté tots aquests objectes i les seves relacions, nodes etc. Aquesta xarxa es comunicarà amb un altre element que gestioni el control de fluxe (és a dir, l'avaluació de l'ordre d'execució de la xarxa), i també podrem serialitzar la topologia de la xarxa en XML.

Capítol 6

Disseny del sistema

Un cop assenyalats els objectius i requeriments del projecte, i la col.lecció d'eines que es necessitaran per a dur a terme la seva realització, és el moment de dedicar-nos a dissenyar i crear el sistema pròpiament dit. Així, aquesta etapa de desenvolupament consistirà en crear un disseny que, tot respectant els requeriments no funcionals, doni resposta als objectius marcats, tot fent servir les eines a la nostra disposició i tenint en compte el domini al qual ens hem d'enmarcar.

Dins aquest disseny hem d'esmentar la part referent a la implementació. És a dir, disseny i implementació formen un tot indissoluble dins un desenvolupament normal dins l'enginyeria de software (i, per extensió, dins l'enginyeria en informàtica).

Mentre es va definint i dissenyant el sistema, o immediatament després, depenent del model de desenvolupament, es creen les seccions de l'aplicació definides al disseny, seguint l'esquema d'aquest.

Depenent del grau de complexitat i detall a què s'arribi en l'etapa de disseny, el temps

dedicat a la implementació será variable també. Això vol dir que teòricament quant més temps duri el disseny, i per tant, més detalls es puguin definir i opcions decidir, menys problemes tindrem alhora d'implementar la solució.

Tot i que això és cert, no podem deixar de banda tota la série d'imprevistos i problemes que inevitablement sorgiran a la llum durant la implementació, i que molt cops fan retardar enormement els temps de creació de sistemes de software.

Per aquest motiu han anat sorgint al llarg dels anys algunes metodologies diferents del típic model en cascada (compost de parts separades en concepte i temps).

6.1 Metodologies de desenvolupament: una visió teòrica

Hi ha tot un seguit de situacions en les que no és recomanable seguir la metodologia en cascada tradicional, segons la qual abans de començar a implementar s'ha de dissenyar tot el sistema, i si alguna cosa és errònia tornar a fer tot el cicle.

En un projecte final de carrera aquest fet és especialment problemàtic, ja que la data d'entrega és fixa i el temps ajustat. Per tant, hauria de ser prioritari aconseguir tenir una base dissenyada i implementada per més tard anar fent iteracions i extenent el sistema, enlloc de fer cada pas del desenvolupament en un bloc unitari.

Així doncs, s'ha triat un model iteratiu de disseny i implementació. Els passos anteriors sí que han estat majoritàriment seqüencials, ja que cada part fa servir l'anterior per a decidir les següents opcions a considerar.

En canvi, dins el disseny i la implementació trobem diverses seccions que es fonamenten unes en les altres, però que no necessiten d'aquest model seqüencial. Com hem comentat, per a crear l'aplicació gràfica de prototipatge necessitem el mode supervisat de CLAM (o alguna cosa semblant), així com també el necessitarem per a definir el sistema de reacTable*.

Per tant, queda clar que el mode supervisat és el que més feina ens portarà i, a la vegada, la part més important.

Aquest mode supervisat es dissenyarà i desenvoluparà de manera incremental, de manera que en qualsevol moment ja hi hagi un sistema (adaptat en major grau als requeriments segons anem avançant) funcionant.

A més, es pretén aplicar alguns aspectes de la col.lecció de metodologies anomenades d'*eXtreme Programming* [25]. D'aquestes idees, dirigides a simplificar els a vegades massa complexos desenvolupaments de programari per tal de fer un codi de millor qualitat i més ràpidament, n'hi ha una de molt poderosa per la seva utilitat: el desenvolupament orientat

a test [5].

Amb la premisa *"Clean code that works"*, l'anomenat TDD (Test-Driven Development) intenta crear implementacions elegants a partir dels requeriments donats, però enlloc d'afrontar-ho com a un tot indivisible ho intenta fer per petites unitats funcionals.

Una iteració comença per fixar petits objectius a resoldre d'aquella unitat funcional, tot escrivint un test a superar pel codi que es faci. És a dir, es pensa qué ha de solucionar el nostre sistema per, tot seguit, pensar un disseny i una implementació que ho resolgui. A més, es busca primerament solucionar el test de la manera més simple possible, per tot seguit crear a partir d'aquesta la versió definitiva.

A més, com que es van creant tests per a totes les seccions del codi, és molt fàcil detectar algun error provocat per futures modificacions: simplement cada cop que s'afegeixin o es modifiquin seccions del codi s'hauran de passar tots els tests que abans funcionaven.

D'aquesta manera s'aconsegueix un codi testejat i exempt de bugs (si més no de la majoria) a cada iteració que es faci, tot afegint les funcionalitats necessàries.

6.2 Aplicació de la metodologia al projecte

El concepte de desenvolupament escollit és intentar crear una xarxa senzilla, però amb la majoria de requeriments ja considerats, i fer evolucionar el sistema de l'estructura més primitiva possible (fent servir el mode no supervisat de CLAM i enllaçant manualment els objectes de processament) fins a un mode supervisat complet.

A través d'aquestes etapes s'aniran dissenyant i implementant (o revisant si ja estan creats) els diferents elements que es van necessitant per a cada fita.

Les restriccions per a aquest sistema són les següents:

- Ús de domini temporal.
- Tamany de buffers de mida fixa
- Topologia de la xarxa sense realimentació ni loops

Com veiem, són majoritàriament les restriccions que tindrà el mode automàtic de reacTable*, amb l'excepció de l'ús d'un sol domini. Tot i això, l'eliminació del domini espectral en aquesta primera etapa ens permetrà concentrar-nos en les seccions més importants (definició de xarxa, flowcontrol, etc.). Així mateix, el segon punt ens elimina la complicació

d'haver de tenir en compte els tamanys dels buffers dels objectes de processament quan s'estableixi l'ordre d'execució.

Finalment, l'últim punt, que com recordem també és un punt de simplificació assenyalat dins l'estructura de reacTable*, permetrà no caure en loops infinits com els comentats a l'etapa d'anàlisi.

Per que fa a la interfície gràfica, es desenvoluparà paral.lelament, i en determinades fites s'establirà un punt d'unió amb el desenvolupament del mdoe supervisat, que pugui mostrar per pantalla el sistema que estigui funcionant en aquells moments. Tot i això, no serà fins a la última etapa que relacionarem completament els dos mòduls.

S'han triat aquestes etapes com a fites del desenvolupament incremental:

1. Diferents configuracions de xarxa funcionant amb el mode no supervisat.
2. Incorporació de l'ús de ports, tot fent servir el mètode d'execució *Do()* en la seva firma de mode supervisat (sense arguments). Les dades seràn guardades en Processing Data directament connectats als ports.
3. Addició de nodes entre ports.
4. Creació de la interfície de xarxa que permeti crear, modificar i eliminar objectes de processament i connexions de port/control.
5. Incorporació de la interfície de passivació i activació XML en la xarxa. Creació d'un FlowControl que treballi amb la xarxa.
6. Mode supervisat complet.

Tenint en compte els diferents casos d'ús que volem estudiar per a crear el mode supervisat, aquestes són les configuracions de xarxa que farem servir al llarg de les 6 fites. El que ens interessa és que cadascuna d'elles sigui capaç d'executar el sistema, i a cada pas que fem ho faci de manera més automàtica.

A més, i tal com hem comentat a l'apartat anterior, s'aniran escrivint tests de funcionalitat, tot seguint la metodologia proposada pel desenvolupament orientat a test, especialment a les seccions més fonamentals del sistema, per assegurar que el codi escrit funciona i no és malmès per futurs canvis.

Tot i això, en la memòria explicarem el disseny dels diferent mòduls i conceptes que formen en el seu conjunt el sistema. És a dir, no seguirem exactament aquestes fites de disseny, sinó que seguirem un ordre que pugui explicar probablement de manera més clara el disseny.

6.3 El camí fins al mode supervisat de CLAM

El sistema que necessitem crear és, en la seva base, un control de fluxe automàtic que permeti modificar la topologia de la xarxa d'objectes de processament i donar suport per a la passivació d'aquesta.

Per tant, haurem de desenvolupar i crear els elements que hem anat comentat al llarg del model del sistema. Tenint en compte que el mode no supervisat de CLAM ja està implementat, la primera cosa que hem d'estudiar és la infraestructura de ports i controls dels objectes de processament.

Aquests elements ens permetran fixar la interfície per la qual entren i surten dades dels Processing, mantenint sempre una visibilitat de l'estructura interna de l'objecte limitada a allò que sigui útil de publicar. És a dir, en cap moment pretenem que es pugui accedir des de fora de l'objecte als seus algorismes de manera intrusiva, de tal manera que siguin elements independents entre ells excepte en aquells punts pels quals sigui possible enviar-los dades.

Després d'analitzar com han de ser aquests ports i controls, s'haurà d'introduir l'ús de nodes (això correspondria al tercer punt del nostre pla de desenvolupament). Aquests nodes introduiràn una automatització de les tasques referents a la gestió de dades entre ports (i, per tant, entre objectes de processament) d'entrada i sortida, així com de la connexió entre ports. Aquest últim punt és delicat, en el sentit que podem tenir diversos objectes que estiguin consumint dades d'un productor, però amb diferents velocitats. Així, s'haurà de tenir en compte coses com la velocitat de lectura/escriptura, el tamany de dades que es produeix/consumeix cada cop, etc.

Un cop dissenyats els sistemes que usin aquests dos passos, és l'hora de crear la xarxa, entesa com el sistema que encapsuli la topologia de connexions entre objectes de processament i nodes, i que a la mateixa vegada pugui oferir la interfície adecuada per a treballar amb ella des de fora (per exemple, des de una interfície gràfica).

Finalment, queda la tasca d'implementar algun tipus d'algorisme que permeti avaluar l'ordre d'execució dels objectes de processament, a fi d'establir un control de fluxe automàtic i eficient.

Pel que fa a la interfície gràfica, s'haurà de dissenyar un sistema que estigui molt desacoblat al *engine* de so (per les raons exposades abans sobre l'estructura en mòduls de reacTable*), però que a la seva vegada sigui capaç d'executar totes les accions que siguin possibles demanar-li a la xarxa des de la vessant de l'usuari (manipulació de connexions, serialització, etc.).

6.4 Nodes

El concepte de node, tal i com l'hem explicat abans, és una de les parts més fonamentals del nostre sistema, en el sentit que és l'encarregat de dur a terme, de manera transparent a l'usuari (i fins i tot al programador d'aplicacions que faci servir el mode supervisat de CLAM) la tasca de controlar les dades produïdes i consumides pels diferents objectes de la xarxa, tot controlant els riscos de sobreescriptura o errades de lectura.

Parlant del concepte, podem dir que un node és una cua de tamany infinit (per tant amb estructura FIFO, *"first in first out"*) en la qual els objectes de processament generadors van dipositant noves dades, que són llegides pels consumidors.

Aquesta idea, que vista des de fora sembla bastant òbvia, es complica en quant anem pensant en què el mecanisme hauria de ser segur i flexible. És a dir, tal i com hem comentat, CLAM no té un sol tipus de dades, sinó que per requeriments en necessita de ben diferents (mostres de so, espectres, segments, etc.). Per tant, els objectes són capaços de produir totes aquestes dades diferents (depenent de la seva funció), i per aquest motiu els nodes han de poder tenir una estructura suficientment general com per a poder suportar aquesta flexibilitat sense fallades ni restriccions degut al seu disseny.

Per altra banda, necessitem un mecanisme segur, en el qual no hi puguin existir errors ni pèrdua de dades importants, fins i tot en un entorn en temps real. Aquest últim punt ens fa veure que a més dels requeriments per a aquest element del nostre sistema, tenim també un altre de no funcional: l'eficiència en la gestió de les dades.

Com a apunt final, l'estructura de nodes ha de ser el més transparent a l'usuari possible, ja que aquest no s'hauria de preocupar en cap mesura per la manera en què les dades viatgen d'un objecte de processament a un altre.

Per a donar solució a tots aquests problemes, s'ha decidit implementar un sistema basat en capes. Perquè aquesta estructura? Doncs bé, hem de tenir en compte que el mode supervisat es caracteritza pel seu automatisme: és a dir, tot el control de fluxe és transparent a qui l'està fent servir. Així doncs, per molta complexitat que pugui tenir l'estructura de nodes (que com veiem pels requeriments no serà trivial), els usuaris han de poder fer-la servir sense complicar-se en entendre tota la lògica que hi haurà a dins, és a dir, tan sols fent servir el control de fluxe i la xarxa haurà de poder fer les operacions que desitgi (fer connexions i desfer-les en temps real, crear nous tipus de dades, etc.).

Passem doncs a desglossar el disseny dels nodes, des de la part més superficial (la interfície) fins a la lògica més complexa del funcionament intern.

6.4.1 La primera capa: la interfície dels nodes

Així, la primera capa, que és la que farà servir qui estigui emprant el mode supervisat, haurà de donar la interfície per la qual els objectes de la xarxa podran produir i consumir les dades que processin.

L'ús dels nodes haurà de ser tan clar com sigui possible. Les operacions bàsiques són demanar-li a un port de sortida que envïi dades als seus consumidors, i per altra banda demanar les noves dades als ports d'entrada. A més, s'han d'incloure mètodes per a poder connectar i desconnectar ports entre ells.

Al diagrama de seqüència 6.1 veiem quina ha de ser la interfície d'accés als ports per accedir a les dades que, com hem comentat, estan regides pels nodes. La xarxa (o qualsevol altre element extern) decideix quins ports es connecten a un node determinat. Quan l'objecte de processament productor de dades n'ha creat de noves, pot enviar-les al port de sortida, i aquest automàticament les diposita al node (amb un mecanisme que explicarem en la següent capa). Aleshores, els objectes consumidors podran demanar les dades als seus respectius ports d'entrada. Aquest mecanisme permet la connexió de més d'un port d'entrada a un sol port de sortida, així com també la possibilitat de connectar-los i desconnectar-los en qualsevol moment, com mostra la figura.

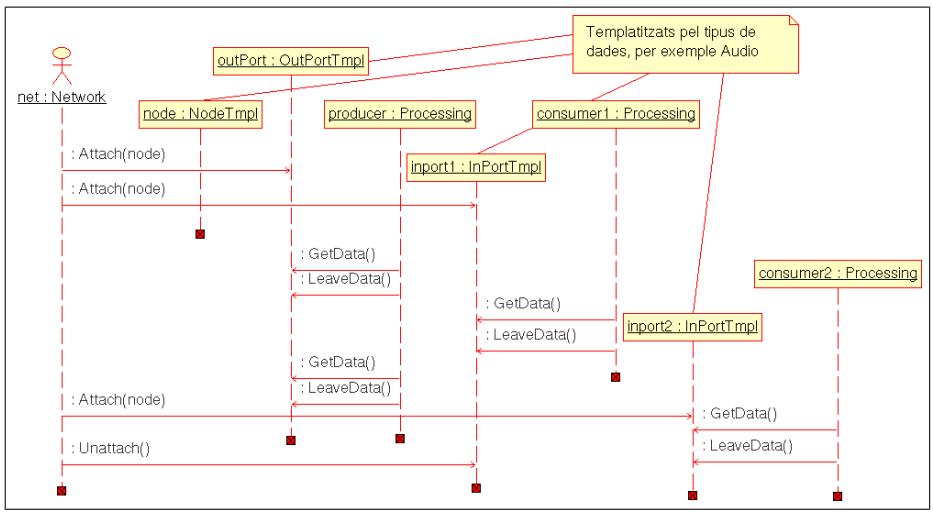

Figura 6.1: Seqüència d'intercanvi de dades en la primera capa de l'estructura de nodes

Hi ha un concepte bàsic per a entendre el funcionament dels nodes: a cada port de sortida li'n correspon tan sols un. És a dir, tan sols tenim un port que vagi produïnt i deixant dades dins el node, ja que per lògica si hi hagués més d'un, les dades produïdes pel primer serien destruïdes pel següent, i els ports que llegeixen no podrien rebre les dades corresponents [1].

[1] Hi ha excepcions com el processament *inplace*, que serà comentat més endavant

Així doncs, tenint això en compte i a partir del diagrama anterior, podem establir el mapa de classes bàsic d'aquesta secció del mode supervisat, a més de les seves relacions. Aquest diagrama és el que podem estudiar a la figura 6.2.

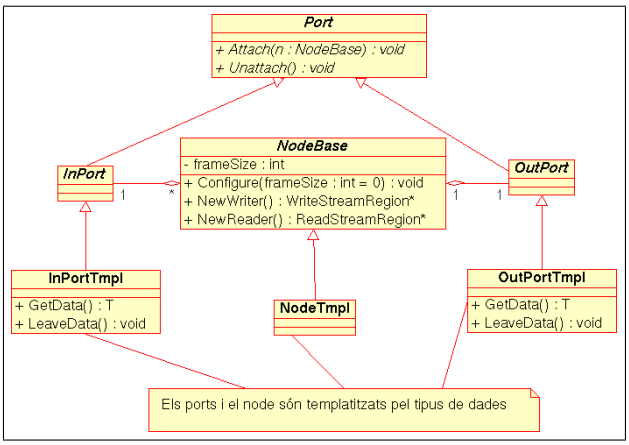

Figura 6.2: Diagrama de classes de la primera capa de l'estructura de nodes

6.4.2 Segona capa: Les regions

Ara que ja hem definit la interfície del sistema, és el moment de fer servir la metàfora explicada abans de la cua infinita per tal de crear la capa immediatament inferior.

A més, un cop vista quina és la relació entre els ports dels objectes de processament i els nodes, hem d'estudiar la manera de solucionar la gestió de la diversitat de dades que podem rebre, així com els diferents tamanys de lectura/escriptura. És a dir, hem de crear un disseny que sigui suficientment flexible per a funcionar eficientment amb diferents configuracions.

A cada port se li ha d'assignar una secció del buffer a on pot fer les operacions que desitgi tenint en compte la seva condició. És a dir, els ports de sortida hauran de tenir una regió d'escriptura en la qual poder deixar les seves dades sense problemes d'escriure a sobre de dades que encara no han estat llegides. Per tant, cap port d'entrada serà capaç de llegir dades de la regió d'un port de sortida que encara no hagi donat la seva autorització per a fer-ho.

Per la seva banda, els ports d'entrada hauran de tenir una regió de lectura en la qual puguin obtenir dades noves, sense perill de repetir dades ja llegides o llegir allà a on encara no ha escrit el port d'escriptura.

La primera imatge que podem extreure d'aquesta relació és que totes les regions de lectura han d'estar per darrera de la regió d'escriptura del port de sortida que, com ja hem comentat

abans, és únic per node. Un exemple de l'estat normal del buffer de dades amb diverses regions seria el diagrama 6.3.

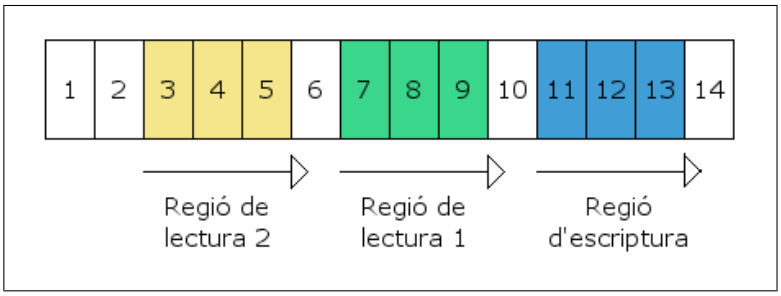

Figura 6.3: Interior d'un node amb dues regions de lectura i una d'escriptura

Així doncs, a fi de mantenir la consistència de les dades processades, abans d'escriure o llegir, les regions hauran de fer comprovacions sobre l'estat de la cua a on estan assignats. Per tant, necessitem una estructura que funcioni com un buffer per sota del node, i que sigui capaç d'anar comprovant que les restriccions es van complint a mesura que es va escrivint i llegint del node. És la classe StreamBuffer. S'encarregarà en tot moment de que les regions de lectura estiguin per darrera de la d'escriptura, i si en algun punt concret hi ha el perill que això no sigui així, podrà indicar-s'ho al control de fluxe.

Això ens porta a parlar de les activacions de les regions. Per tal que una regió sigui capaç de llegir o escriure, s'haurà d'*activar*. Aquesta activació la du a terme l'StreamBuffer, tot tenint en compte el coneixement que té de la situació de totes les regions dins ell. Aquest pas d'activació vol dir que la regió comprovarà que és possible llegir/escriure sense infringir cap de les restriccions, i aleshores permetrà al port obtenir o deixar les dades.

Com veiem, per a mantenir la metàfora de buffer infinit esmentada en la primera capa, hem d'introduir més elements dins la dinàmica d'intercanvi de dades. Fent servir com a exemple la seqüència vista abans (però simplificada a una sola regió de lectura), aquests serien els mecanismes d'interacció de dades, incorporant ara la part de regions i la classe StreamBuffer. 6.4. Tot i que no s'han posat els paràmetres per no fer massa llarg el diagrama, l'StreamBuffer té sobrecàrregues dels mètodes *GetAndActivate* i *LeaveAndAdvance*, depenent de la regió que es passi com a paràmetre (de lectura o escriptura, inplace, etc.). D'aquesta manera podrà saber quines comprovacions dur a terme, segon el tipus de regió que es vulgui activar.

Aquest disseny, en què les diferents regions tenen restriccions a complir, ens permet introduir diversos tamanys de finestra de lectura/escriptura (que en definitiva vol dir regions amb tamanys diferents), depenent de les dades que necessiti un objecte de processament en un moment determinat.

Per a concloure l'estudi d'aquesta segona capa, aquí tenim el diagrama de classes donat per

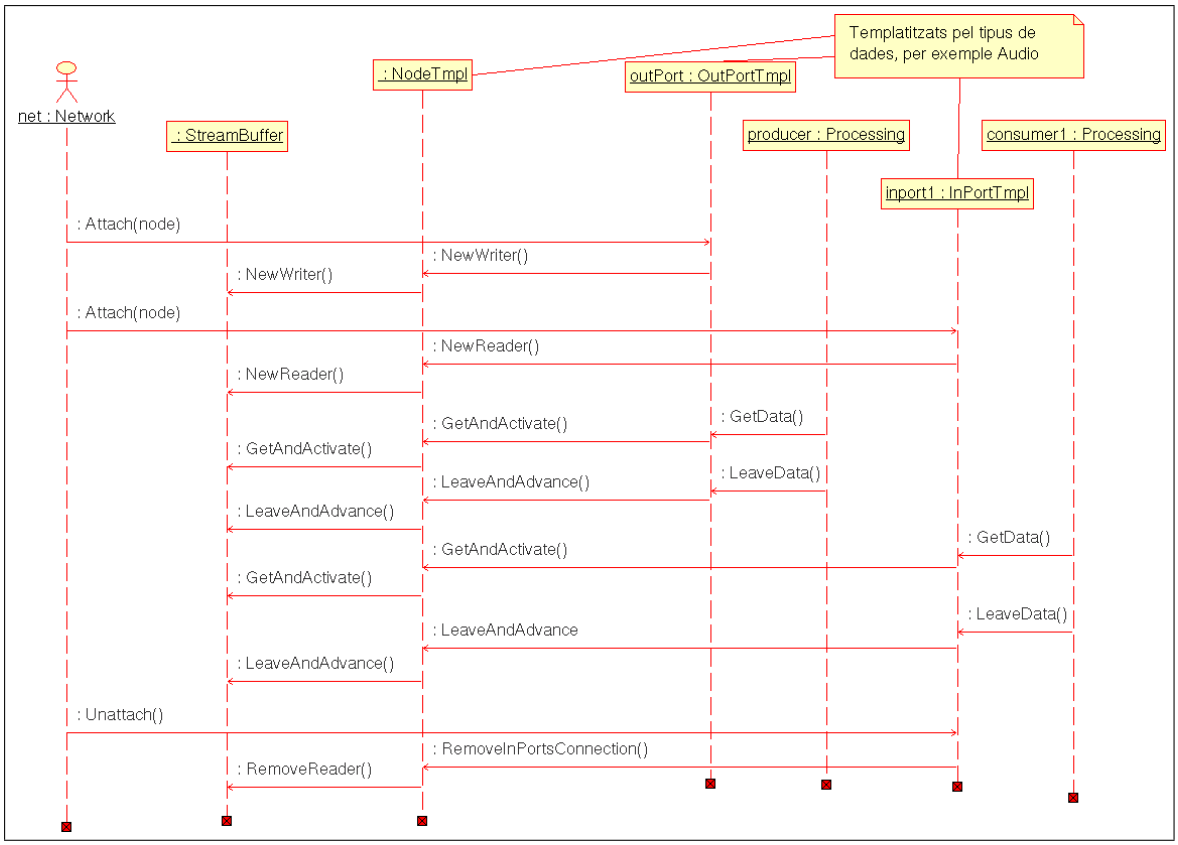

Figura 6.4: Seqüència d'intercanvi de dades en la segona capa de l'estructura de nodes

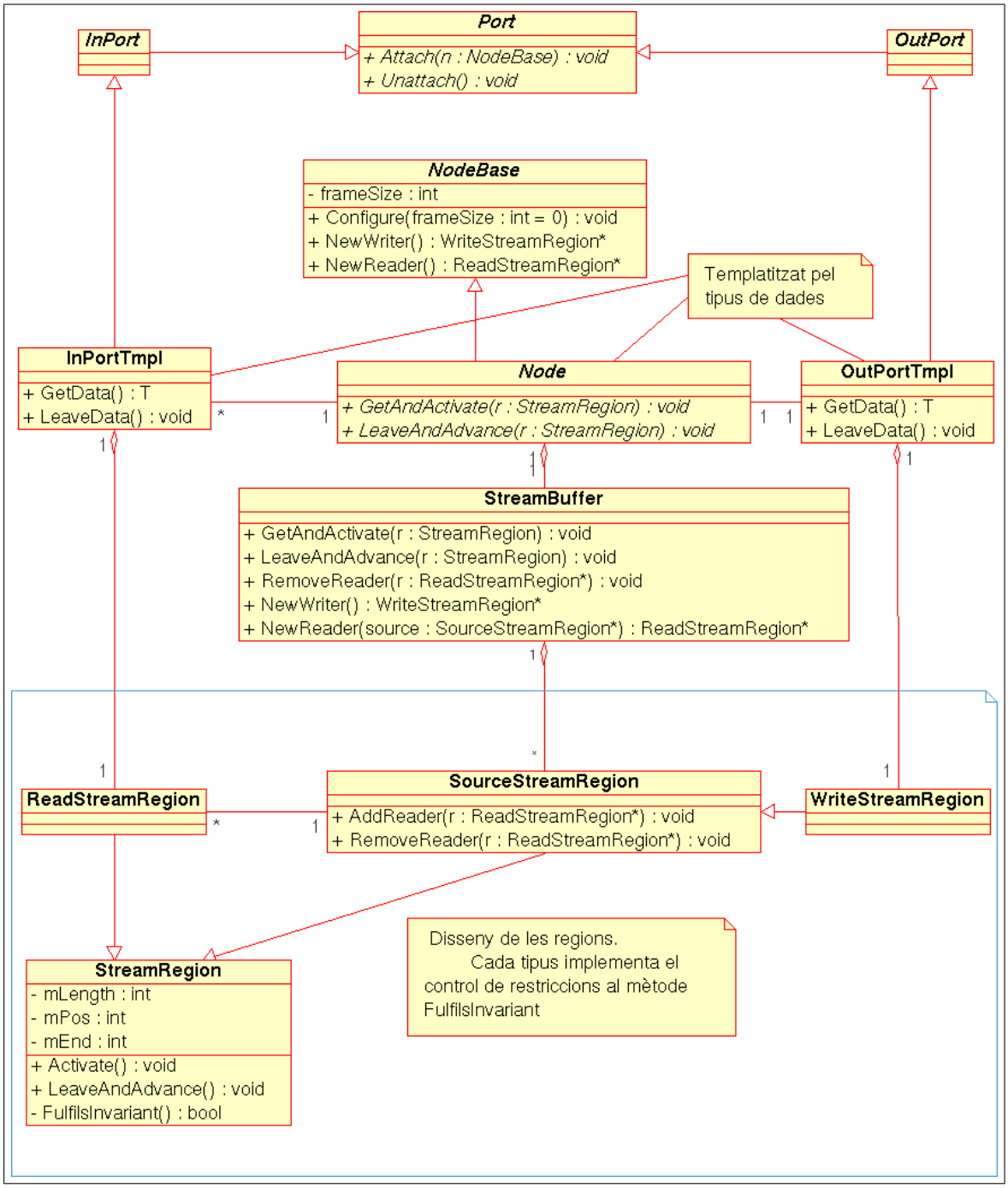

Figura 6.5: Diagrama de classes de la segona capa de l'estructura de nodes

la interacció entre objectes de la figura anterior. Com veiem el disseny de les regions fa que puguem tenir diversos tipus de lectors i escriptors (segons les necessitats), i tots els ports connectats a un node hauran de registrar les seves regions al contenidor que té, com podem veure aquí. 6.5

6.4.3 Tercera capa: La implementació de la cua

Així doncs, ja hem solucionat la forma de gestionar diversos tipus de regions, fins i tot amb tamanys diferents. Ara ens queda el problema de resoldre la lògica de la cua infinita, ja que per definició és impossible emular això en un ordinador (per les restriccions de la memòria, que és finita). Per tant, s'hauran de crear mecanismes que simulin aquesta metàfora amb alguna implementació que, per altra banda, ha de ser com més eficient millor.

Com que hi ha diverses maneres de resoldre aquest problema, s'ha optat per crear una estructura semblant al patró de disseny Estratègia. Així, quan es crea un node, a més de templatitzar-ho pel tipus de dades com hem comentat abans, també es templatitzarà pel tipus d'estratègia a seguir en la implementació del buffer. D'aquesta manera desacoblem les dues primeres capes amb aquesta tercera, ja que en qualsevol moment podrem canviar l'algorisme encarregat de la lògica del buffer sense haver de modificar en cap moment els ports ni les regions.

Per ara s'ha implementat l'algorisme més usual en aquest tipus de situacions: un buffer circular. Així, un cop recorregut tota la cua, les regions tornen a passar per les primeres posicions del buffer, tot emulant la cua infinita, que és el que necessitem. Això ens comporta un altre problema a resoldre, i és el tamany de les regions. És a dir, com que una regió normalment tindrà un tamany superior a 1, haurem de controlar qué passa quan les regions arriben al final del buffer, ja que teòricament hauran de llegir dades al final del buffer circular i també al principi, com ens mostra la figura 6.6

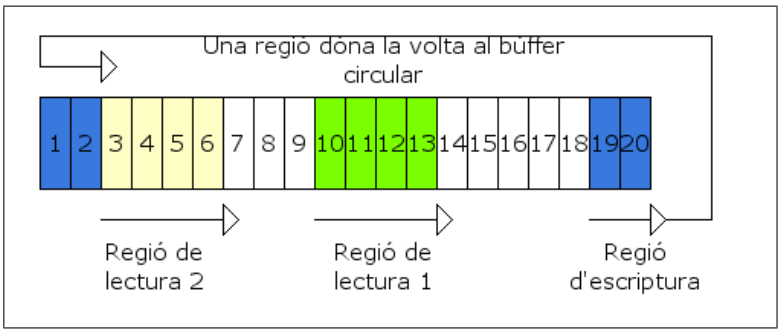

Figura 6.6: Problema de la implementació de nodes amb búffers circulars

La solució és crear una *zona fantasma* dins el buffer. Reservem un espai al final del buffer

tan gran com la major regió de totes les que hi ha, i fem d'aquesta regió un duplicat del que hi ha a l'inici del buffer. Així, si una regió està a cavall del final del buffer i l'inici, llegirà (o escriurà) a la zona fantasma, i després es passarà això a les primeres posicions del buffer. D'aquesta manera no és necessari partir les regions quan arribin al final del buffer. Al diagrama 6.7 veiem un gràfic de l'estructura del buffer circular, inclosa la zona fantasma.

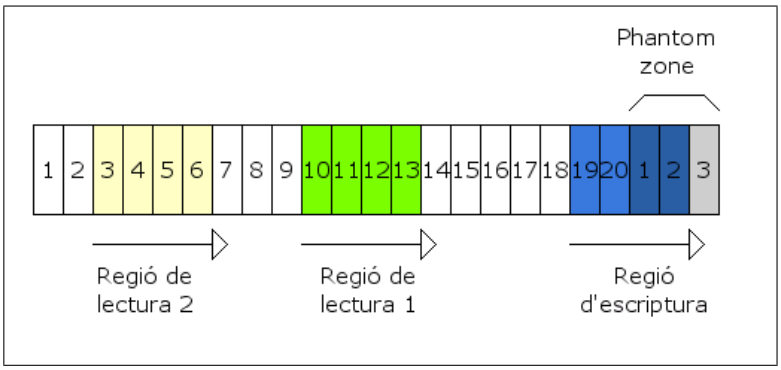

Figura 6.7: Solució amb l'ús d'una zona fantasma

D'altra banda, aquest buffer circular haurà de controlar les restriccions derivades de no tenir una memòria física infinita.A més de controlar que no se solapi cap regió de lectura amb la d'escriptura tenint en compte la lògica de la zona fantasma, de cap manera ha de permetre que una regió d'escriptura que processi més ràpida que les de lectura doni la volta al buffer, de tal forma que sobrepassi a la última regió de lectura.

Per tant, tenim que cada capa controla les restriccions imposades per ella mateixa en quant a sobreescriptura i solapament, i que per quan es cridin als mètodes de la primera capa per a deixar o recollir dades dels ports, aquests es comunicaran amb el node corresponent, es farà un seguit de comprovacions per saber si és possible l'acció (tot aprofudint en les capes per a mirar que tots els nivells mantinguin la integritat), per finalment dur a terme el procés de lectura/escriptura de les dades.

Al diagrama de seqüència 6.8 veiem totes les interaccions entre objectes que genera les crides inicials a la primera capa. S'ha eliminat el port d'entrada perquè la seqüència de crides ampliada (la de recollir dades) és idèntica a la del port de sortida. És la classe CircularStreamImpl qui sap de quin tipus és la regió, i per tant llegirà o escriurà segons convingui.

Una altra punt a comentar, relacionat amb l'extensibilitat del sistema, és que l'estructura de regions actual permet crear-ne de noves, com per exemple regions *inplace* que llegeixen una secció de la cua i a continuació hi tornen a escriure, o també regions amb *delay* que llegeixin dades endarrerides respecte a la seva posició.

Al diagrama de classes 6.9 observem les relacions entre els diferents elements del sistema

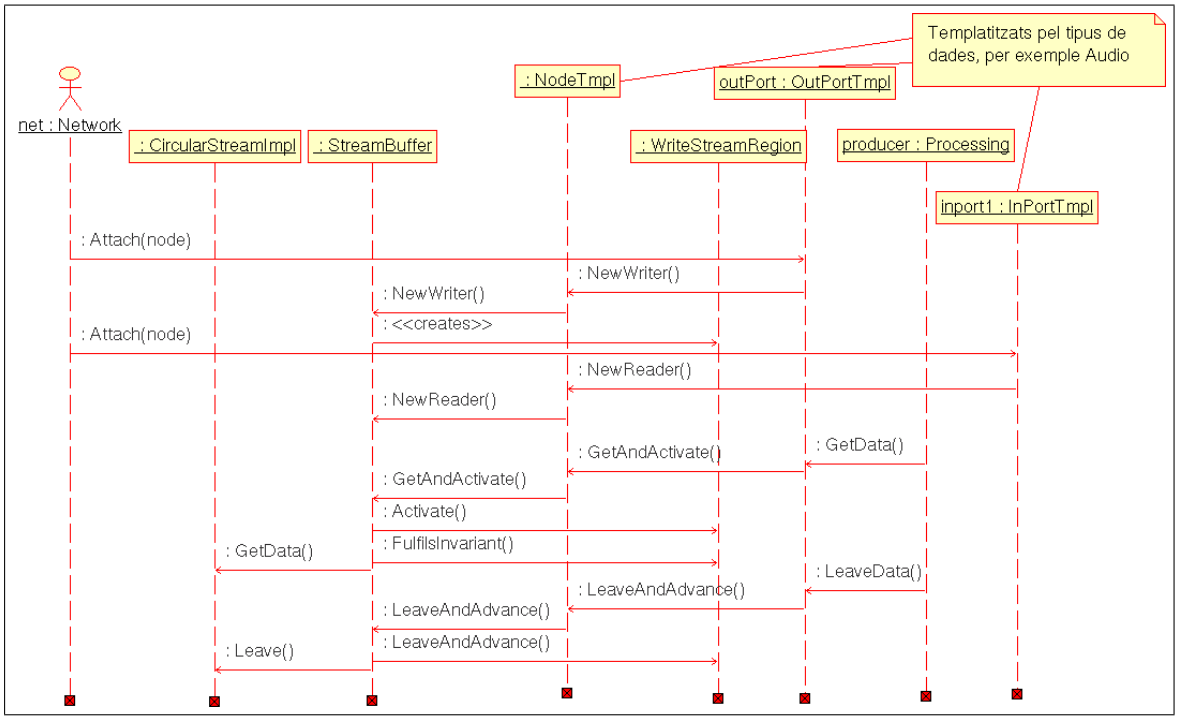

Figura 6.8: Seqüència d'intercanvi de dades en la tercera capa de l'estructura de nodes

d'intercanvi de dades del fluxe de control, inclosos ja els elements d'implementació de la cua.

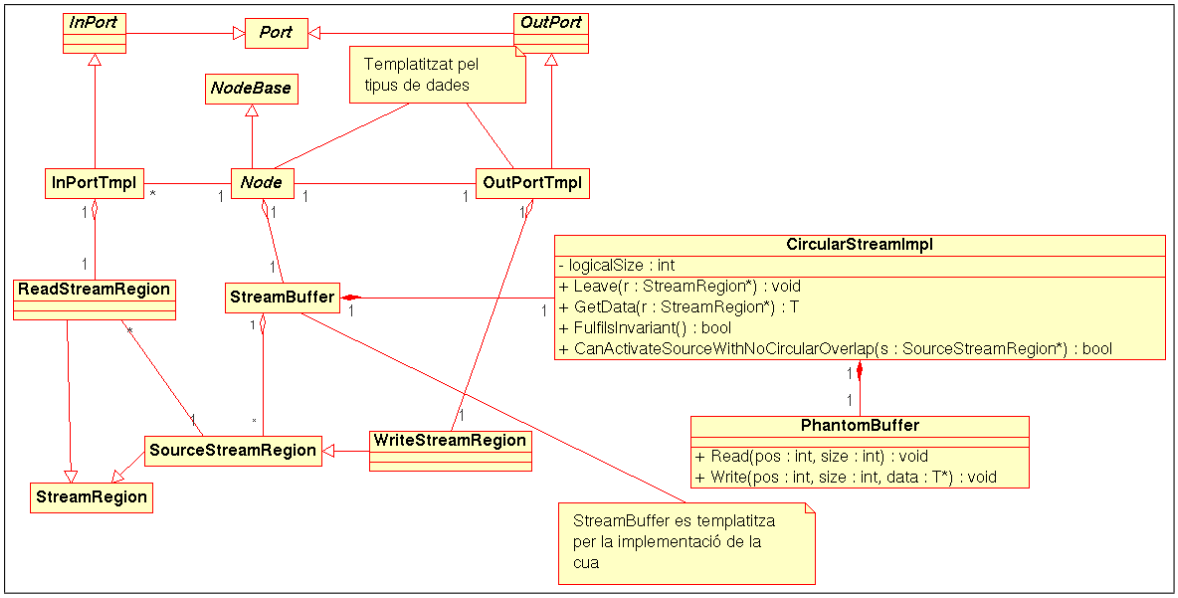

Figura 6.9: Diagrama de classes de la tercera capa de l'estructura de nodes

6.5 La xarxa de processament

Un cop definit quin ha de ser el disseny per la capa bàsica del control de fluxe (és a dir, com s'han d'intercanviar les dades entre els objectes de processament), ja es pot començar a crear el nivell que l'usuari farà servir: la xarxa de processament.

Aquesta xarxa, els requeriments de la qual hem estat comentant a l'apartat de requeriments i model del sistema, té una interfície inicial bastant clara, i el disseny haurà de permetre fer-la servir de tal manera que respongui als canvis en la topologia de la xarxa de manera automàtica, tot fent servir els nodes i ports per a gestionar els intercanvis de dades sense caure en incoherències (llegir de ports quan ja estan desconnectats, accedir a objectes de processament ja eliminats, etc.).

Per tant, aquesta és la interfície bàsica que s'ha de procurar a l'usuari per tal de modificar la topologia de la xarxa:

- Afegir/Borrar un objecte de processament.

- Crear/Eliminar una connexió entre ports d'objectes de processament ja creats.
- Connectar un control de fluxe a la xarxa.
- Engegar/Aturar el processament de dades.

A més d'aquests mètodes, serà necessària una interfície per a la passivació i activació de la xarxa desde disc dur. Per a poder facilitar l'ús d'aquesta interfície, s'ha de pensar alguna manera còmode per a poder explicitar quins són els objectes de processament i ports sobre els quals volem aplicar les diferents accions.

6.5.1 Propietat dels objectes de processament

Per definició, la xarxa hauria de tenir la propietat de tots els nodes i objectes de processament que en formin part. Això vol dir que, un cop els ha afegit, l'usuari s'ha de desentendre d'ells, ja que la xarxa haurà de gestionar les modificacions que se'n facin, així com també la seva destrucció quan ja no siguin útils.

Així doncs, quan vulguem accedir a l'objecte, tan sols hem de comunicar el nom de l'objecte desitjat (que és únic dins la xarxa) La forma d'accedir als diferents ports ha de ser similar: l'úsuari no ha de conéixer exactament els ports a connectar, tan sols el nom del port i el nom de l'objecte propietari d'aquest. D'aquesta forma, tindrem una interfície d'accés intuïtiva amb crides com *network.ConnectPorts("ObjecteProductor.PortDeSortida", "ObjecteConsumidor.PortDentrada").*

6.5.2 Comunicació amb l'algorisme de control d'execució

Per a mantenir la transparència de tot el moment, és lògic pensar que l'usuari hauria d'accedir tan sols a una interfície per tal de fer els canvis que desitgi en la xarxa, tant en la topologia com en l'estat de processament (és a dir, engegar-la, aturar-la, etc.).

L'alternativa seria tenir dues interfícies, i comunicar-nos per una banda amb la xarxa quan vulguem modificar la topologia amb els mètodes anteriors, i per l'altra haver d'accedir a l'algorisme de control de l'execució quan s'hagi de modificar l'estat del processament. Com es pot deduir, això tan sols serveix per complicar la vida a l'usuari, si tenim en compte que realment són dos conceptes que van lligats (l'ordre d'execució no té sentit si no és dins una xarxa).

Resumint, es pot pensar que l'algorisme d'ordre d'execució és una *estratègia* emprada per la xarxa per a dur a terme les crides que fa l'usuari relacionades amb l'estat, és a dir:

- Configurar els objectes de processament i els nodes entre ells per a l'execució.
- Executar una iteració de tots els objectes de la xarxa.
- Opcionalment, fer algun tipus de precàlcul per tal d'establir l'ordre de manera eficient.

Per tant, es segueix el patró de disseny (que també s'ha usat en la secció de la implementació dels nodes) per a establir la relació entre la xarxa i l'algorisme, com veiem al diagrama 6.10

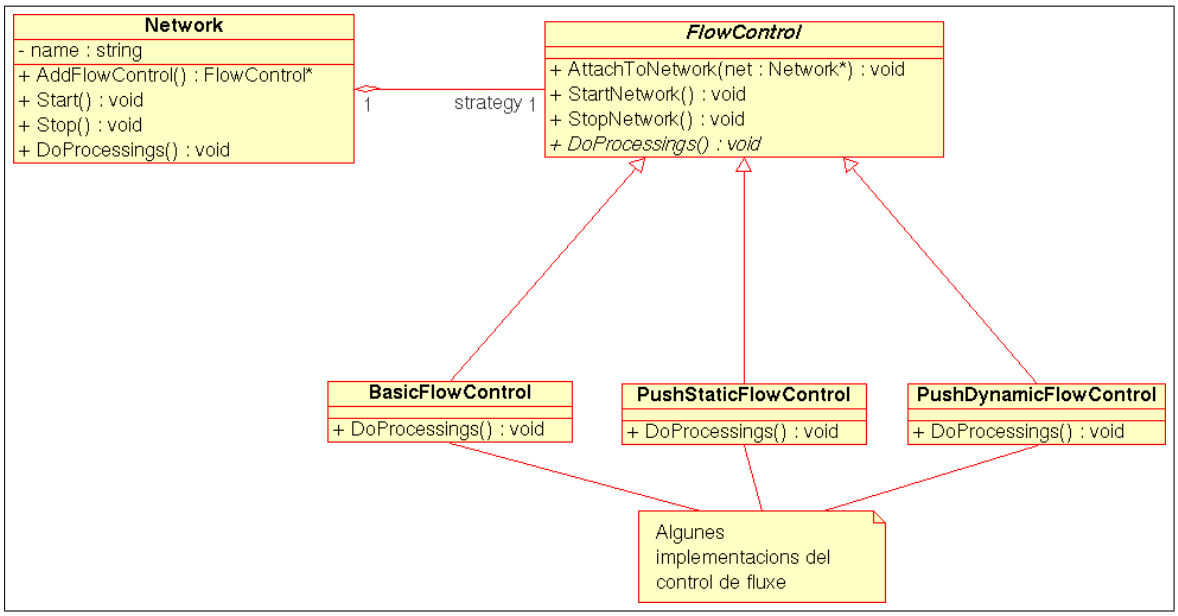

Figura 6.10: Patró de disseny estratègia usat a la Network

6.5.3 Serialització

A més dels mètodes bàsics per a modificar la xarxa, també són necessaris, com s'ha explicat als requeriments, poder desar una xarxa a un fitxer per a recuperar-ne la topologia posteriorment.

Per a fer-ho, necessitem dos components principals. Per una banda, algun mètode per a automatitzar la passivació i activació per a qualsevol tipus de xarxa. Per a fer això, podem fer servir una llibreria de suport a serialització. Les alternatives són clares: com que la xarxa ha de poder-se editar des de fora d'una aplicació explícitament dedicada a això, no pot ser emmagatzemada en format binari, sinó que s'haurà de desar en codificació entendible per l'usuari, per exemple ASCII. D'aquesta forma es podrà modificar la xarxa

amb qualsevol editor de textos. A més, s'ha de donar una estructura clara del document a fi que qui l'estigui modificant sàpiga trobar el que vol canviar de la xarxa. Per tant, una solució òbvia és l'ús de XML i una llibreria per a crear-lo, com per exemple Xerces. A més, així es mantindria la integritat amb la resta del framework CLAM, que la fa servir per a la passivació de dades i configuracions.

L'altre component és algun mecanisme de creació d'objectes de processament. Aquest mecanisme hauria de permetre crear qualsevol tipus d'instància de *Processing*, de tal manera que l'estructura de xarxa no hagi de conéixer totes les classes d'objectes, cosa que serial realment incòmode i poc extensible, en el sentit que aleshores al crear un nou tipus d'objecte de processament, hauríem de modificar la classe Network. Per tant, hem de crear un disseny en el qual diferents tipus d'objectes, tots derivats d'una classa base comuna (és a dir,*Processing*), han de ser creats per una entitat que no coneix els tipus de classes filles i que, a més, aquestes poden variar (tot afegint-ne de noves).

Per sort aquest tipus de problema és comú a moltes aplicacions, i per tant hi ha una resposta concreta. És el patró de disseny anomenat *Factory Method*, que podem veure al diagrama 6.11.

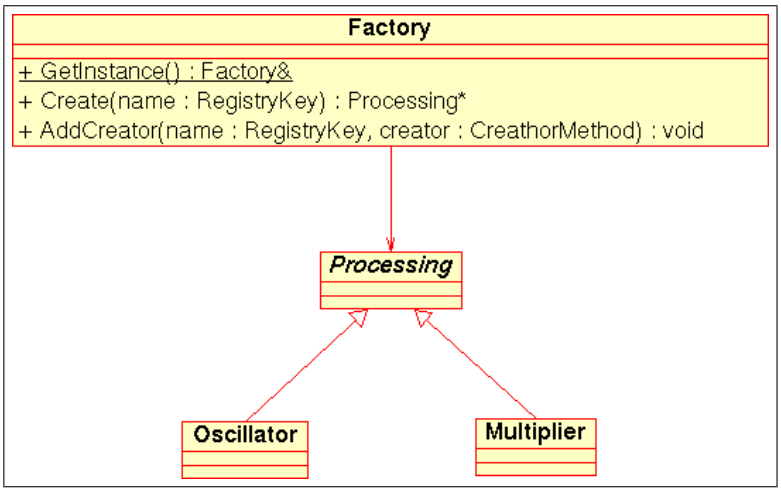

Figura 6.11: Patró de disseny Factory Method per a crear nous objectes de processament

Com veiem, es delega la creació d'objectes a una "fàbrica" en la qual hem registrat abans els mètodes que permeten crear els tipus de classes desitjats.

Per tant, el mecanisme per a activar una xarxa a partir d'un fitxer en XML que en defineixi la topologia és, tot fent servir una *Processing Factory* amb els diferents constructors ja registrats, crear els objectes que la componien amb la configuració corresponent que hem desat a disc, per després fer servir la interfície d'accés comentada abans per a crear les connexions necessàries entre aquests objectes. D'aquesta manera podrem recrear la topologia

passivada.

6.6 Ordres d'execució

Com hem comentat abans, l'usuari decideix l'estratègia a sequir en quant a l'ordre d'execució dels objectes de processament en una xarxa. Oferint una interfície comuna, la classe abstracta FlowControl permet desentendre's del tipus de topologia i les configuracions necessàries dels nodes i els objectes de processament, i centrar-nos en l'algorisme per a decidir com els executem. Això es pot decidir usant des d'algorismes trivials fins a *schedulers* que tinguin en compte els tipus de dades, variacions de tamany de les regions, etc. El disseny permet aquesta diversitat d'algorismes, tot procurant virtualitat en tots els mètodes de FlowControl per si és necessari redefinir-ne algun (per exemple al engegar la xarxa, al afegir un objecte, etc.).

Podem veure una xarxa de processament com una espècie de graf dirigit de connexions d'un autòmat finit, en el qual podem trobar llaços, divergència de camins de les dades, etc. Per tant, els algorismes teòrics de recorregut de grafs ens poden servir com a base per a crear-ne de propis i eficients (mètodes de push, pull, etc.).

6.6.1 Algorisme d'exemple

Com a exemple hem triat un dels algorismes comentats a l'article ARTICLE DE PUSH, basat en una cerca en amplitud seguint un mètode de *pushing*. Això vol dir que es basa en començar pels estats inicials del graf i anar avançant a través dels nodes als que puguem arribar.

Traduït al nostre domini, l'algorisme començarà pels objectes de processament que no necessitin consumir dades d'altres objectes, és a dir els generadors (per exemple, oscil.ladors, reproductors de fitxers d'entrada, etc.). A partir d'aquí, anirem recorrent la topologia de la xarxa sempre en sentit *productor -¿ consumidor*, fins a acabar d'executar tots els objectes disponibles.

L'algorisme dissenyat vé donat per aquestes accions:

```
conjunt PerFer = objectes generadors;
conjunt Fets = buit;

Mentre PerFer no sigui buit
{
```

```
  Treure un objecte X de Fets;
  Executar X;
  Guardar X a Fets;

  Per cada objecte Y fill de X
  {
     Si tots els pares de Y estan a Fets,
          Guardar Y a PerFer.
  }
}
```

6.7 Visualització

Paral.lelament al desenvolupament del control de fluxe del sistema s'ha anat creant l'aplicació gràfica que ha de servir per a modificar la topologia de la xarxa i permetre a l'usuari fer servir l'infraestructura dissenyada per a automatitzar la gestió de dades entre objectes de processament.

Un dels requeriment més important de la GUI, com ja hem esmentat abans, és el baix acoblament que ha de tenir amb l'*engine* de so. Això vol dir que tant els elements de la xarxa de processament com l'estructura dels objectes de processament no han de saber que hi ha una interfície gràfica connectada a ells. Així, s'haurà de dissenyar una aplicació que, mitjançant la interfície de la xarxa que abans hem comentat, sigui capaç de dur a terme tots els casos d'ús especificats a la secció de requeriments del sistema.

Afortunadament, el framework CLAM ja té un mòdul de visualització en funcionament dissenyat específicament per a complir aquests requeriments, és el que s'exposa a [40].

Per tant, podrem fer servir les idees d'aquesta solució, tot i que adaptant-les a les necessitats pròpies del sistema que s'està desenvolupant. Aquestes s'allunyen una mica dels requeriments stàndard del mòdul de visualització, ja que en cap moment haurem de presentar dades a l'usuari. Enlloc d'això haurem de ser capaços d'extraure la topologia de la xarxa, i la informació relativa als objectes de processament que necessitem (número de ports i controls, nom, si estan connectats o no, etc.).

6.7.1 El mòdul de visualització de CLAM

El mòdul de visualització de CLAM fa servir el patró anomenat Model Vista Controlador per a desacoblar la secció d'interfície gràfica dels elements que volem observar. L'estructura general del patró es pot veure al diagrama de classes 6.12.

Bàsicament intenta dur a terme aquest desacoblament fent servir una capa intermitja, anomenada vista, entre l'objecte a ser observat (és a dir, el model) i la seva presentació en pantalla (el controlador), amb la finalitat de fer el tractament de les dades necessari per a exposar-ho a l'usuari en un element que no sigui el model en sí.

Aquesta vista s'encarregarà de mantenir actualitzades les dades necessàries del model, i passar-se-les al controlador. De la mateixa manera, quan l'usuari modifiqui algun aspecte de les dades en pantalla, el controlador ho notificarà al model.

Les avantatges d'aquest sistema són clares, ja que aconseguim que el model no conegui en cap moment que està essent observat per un element gràfic, i que, per tant, qualsevol

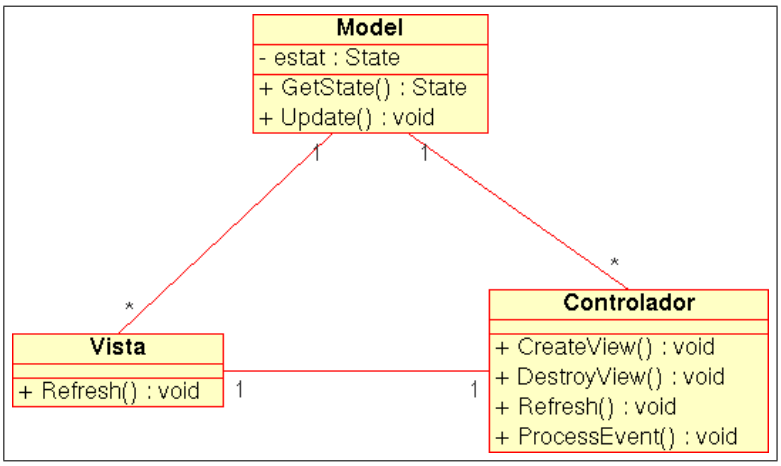

Figura 6.12: Diagrama de classes del Patró de disseny Model-Vista-Controlador

modificació en la secció GUI no l'afecti. Això es pot observar en el diagrama de seqüència 6.13, en què podem observar un intercanvi de dades teòric entre les diferents capes del patró.

A més, es resol així un altre objectiu de la GUI, que era la multiplicitat d'aplicacions gràfiques usuàries del nostre sistema. Com ja hem comentat abans, a part de l'aplicació per a modelar xarxes de processament, aquest mode automàtic de processament de so serà emprat al projecte reacTable*, que visualitzarà les dades mitjançant projectors de vídeo i en rebrà les modificacions dels usuaris amb càmeres i tècniques de reconeixement de formes.

Amb el sistema MVC, tan sols haurem de crear de nou la capa de controlador (és a dir, una per a la aplicació de modelatge i una per a reacTable*). La capa de vista serà exactament la mateixa. Això també ens estalviarà modificar el disseny si en qualsevol moment es decideix crear una versió amb un altre toolkit gràfic.

L'aplicació d'aquest model dins CLAM ens porta a considerar alguns problemes del patró de disseny ja apuntats al treball realitzat per Miguel Ramírez. Un dels principals és que, tot confiant en l'actualització de les dades des del controlador, tenim un accés doble (per part de la vista i per part del controlador) al model. Aquest problema pot ser realment complicat de resoldre si pensem en entorns multifil i programació concurrent, d'altra banda molt típic en aplicacions de música en temps real; un fil per a la interfície gràfica i un per a l'engine de so, que treballen a velocitats molt diferents.

A més, el model hauria de tenir els mètodes necessaris de refresc, cridats pel controlador.

Per tant, l'aproximació adoptada per CLAM és lleugerament diferent. El controlador (anomenat ara presentació) no ataca la interfície del model, sinó que tan sols es comunica amb la vista. Per tant, aquesta es divideix en dues classes, que es faran servir depenent de

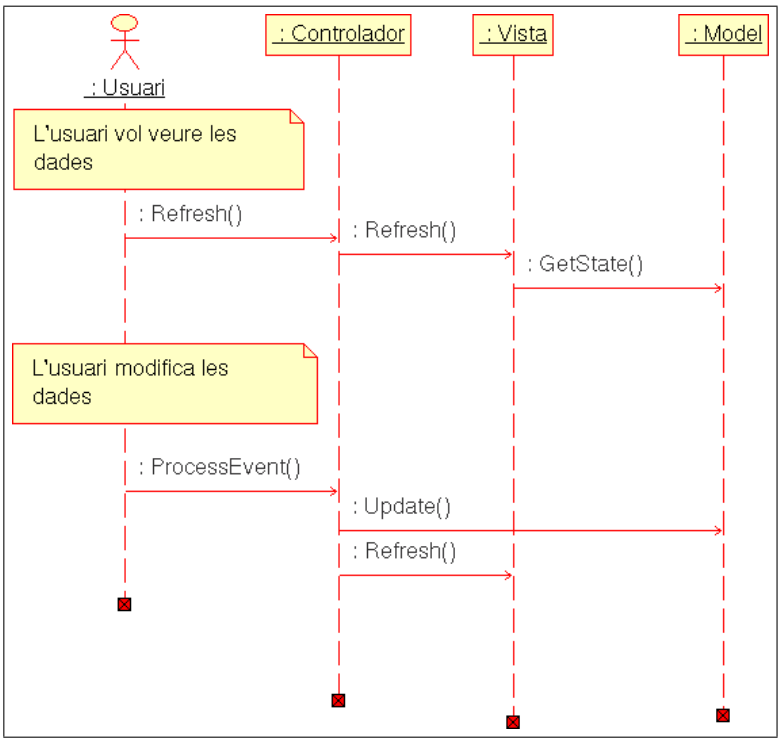

Figura 6.13: Diagrama d'interaccions entre els elements del patró Model-Vista-Controlador

l'ús que se li vulgui donar a la GUI. Si és una aplicació gràfica que tan sols vol presentar dades sense modificar-les, s'usarà la classe Adapter (que bàsicament implementa el concepte de Vista anterior). Si, per contra, també es vol modificar el model, es farà servir la classe Controller, que té dret a modificar alguna secció del model.

D'altra banda, s'ha resolt l'intrusisme del patró al model amb una altra classe anomenada Model, que és la que coneix realment a l'element a ser observat. La Vista concreta que es necessiti (ja sigui Controller o Adapter) deriva tant de Controller o Adapter (classes abstractes que donen la interfície que es necessita) com de la classe Model concreta.

Resumint, tenim una classe que pot observar i modificar el model a través d'un *stub*, i que, per tant, no necessita conéixer les dades de manera intrusiva. A més, tindrem una presentació concreta, que es comunicarà a través del Controller/Adapter amb el model, i que serà la que sigui implementada amb un toolkit gràfic (o qualsevol altra forma de visualització).

6.7.2 Disseny de l'aplicació gràfica

Un cop decidida l'estructura general del sistema i com resoldre els problemes que se'ns han plantejat als requeriments, és hora de concretar el disseny.

Seguint el disseny del mòdul de visualizatció, tenim una estructura en la qual no hi ha intrusisme al model, com es pot veure al diagrama de classes 6.14 en la relació entre Controller/Adapter i el model.

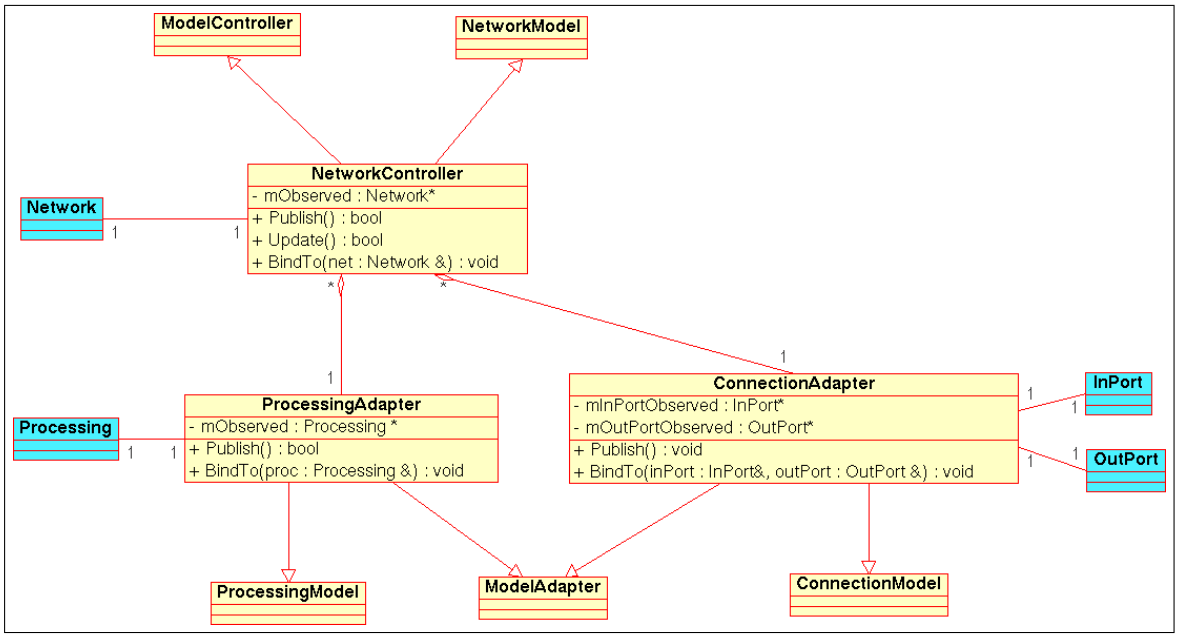

Figura 6.14: Diagrama de classes d'un controlador de Network

Començant per la capa més general de representació la de la xarxa de processament, podem veure que bàsicament el que hem de representar seran els diferents objectes de processament i connexions que conté. Per tant, l'estructura general per a la representació del sistema és un model de *composite*, en què cada element contingui, a més de les dades que facin falta, les vistes de les instàncies contingudes, en una estructura jerarquitzada.

L'estructura en arbre del patró *composite* es pot veure dins l'adaptador de la classe Processing 6.15 , a on es repeteix la mateixa estructura aproximadament que al diagrama anterior.

Així, estarem replicant l'estructura de la xarxa en una altra de vistes i una altra final de presentacions. Tot i que això pugui comportar una complicació del disseny general, seria pitjor crear una sola capa amb tota la informació, ja que seria bastant poc intuïtiva, i canviar alguna cosa del disseny seria realment complicat. D'aquesta manera solucionem un

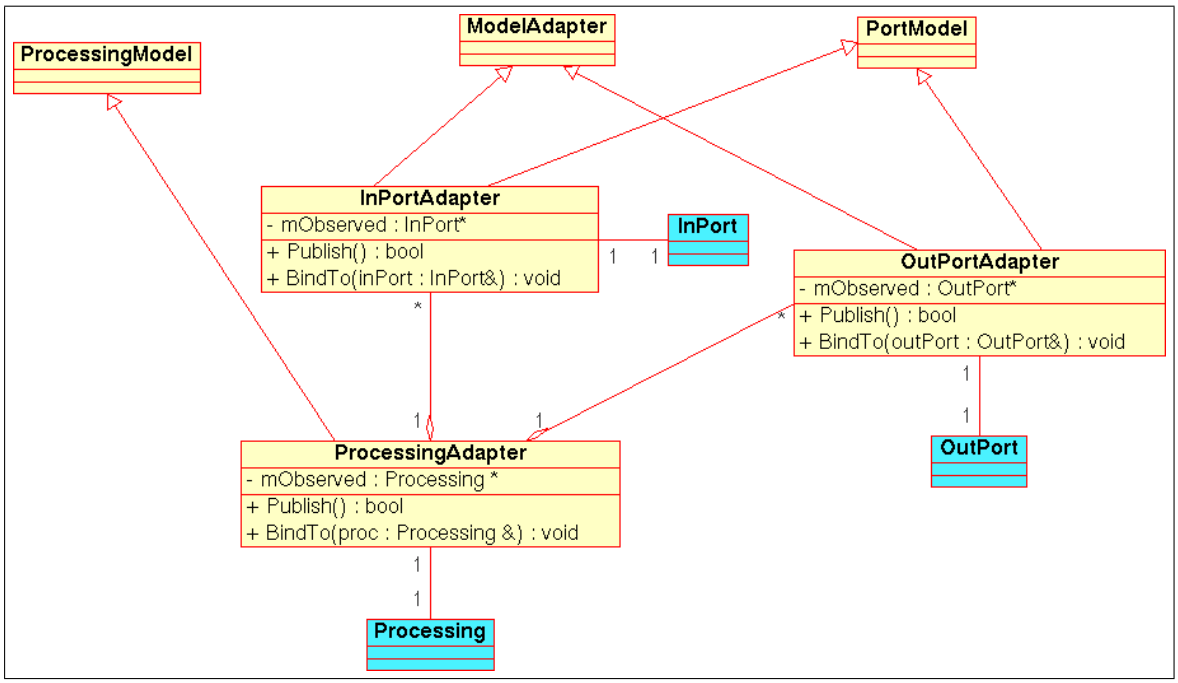

Figura 6.15: Diagrama de classes de l'adaptador d'un objecte de processament

teòric problema de cohesió, ja que cada classe té ben delimitades les seves responsabilitats.

6.7.3 Implementació de la presentació

Un cop definit el mecanisme general de l'aplicació, s'ha dissenyat i implementat la última capa fent servir la llibreria gràfica QT, per les raons explicades en l'apartat d'eines. Així, després de dissenyar les capes generals que seran emprades tant per la GUI de l'aplicació que he de crear, com per reacTable*, tan sols fa falta crear les presentacions corresponents sota el toolkit que es vulgui.

6.8 Resultats

Per tant s'ha aconseguit crear un disseny que compleix tots els requeriments d'eficiència, transparència i desacoblament entre mòduls que se'ns ha donat al principi. Així, un usuari que vulgui fer servir el mode supervisat de CLAM (o reacTable*, per exemple), tan sols haurà d'anar modificant les connexion entre objectes de processament, així com afegir-ne

de nous o eliminar-ne algun. El sistema automàticament farà les notificacions oportunes a la xarxa, que modificarà al seu torn la topologia del graf de connexions que tingui. Com veiem és un model molt potent, a més de simple des del cantó de l'usuari, ja que aquest no ha de conèixer com funciona la lògica dels nodes. De fet, si no vol no ha de saber ni que existeixen, ja que és la xarxa qui s'encarrega de crear-los i afegir lectors o escriptors, depenent de la tasca que se li demani desde la interfície d'usuari.

Podem veure un exemple de xarxa en aquesta captura de pantalla, a on veiem diversos objectes de processament i les connexions entre ells dins una xarxa 6.16

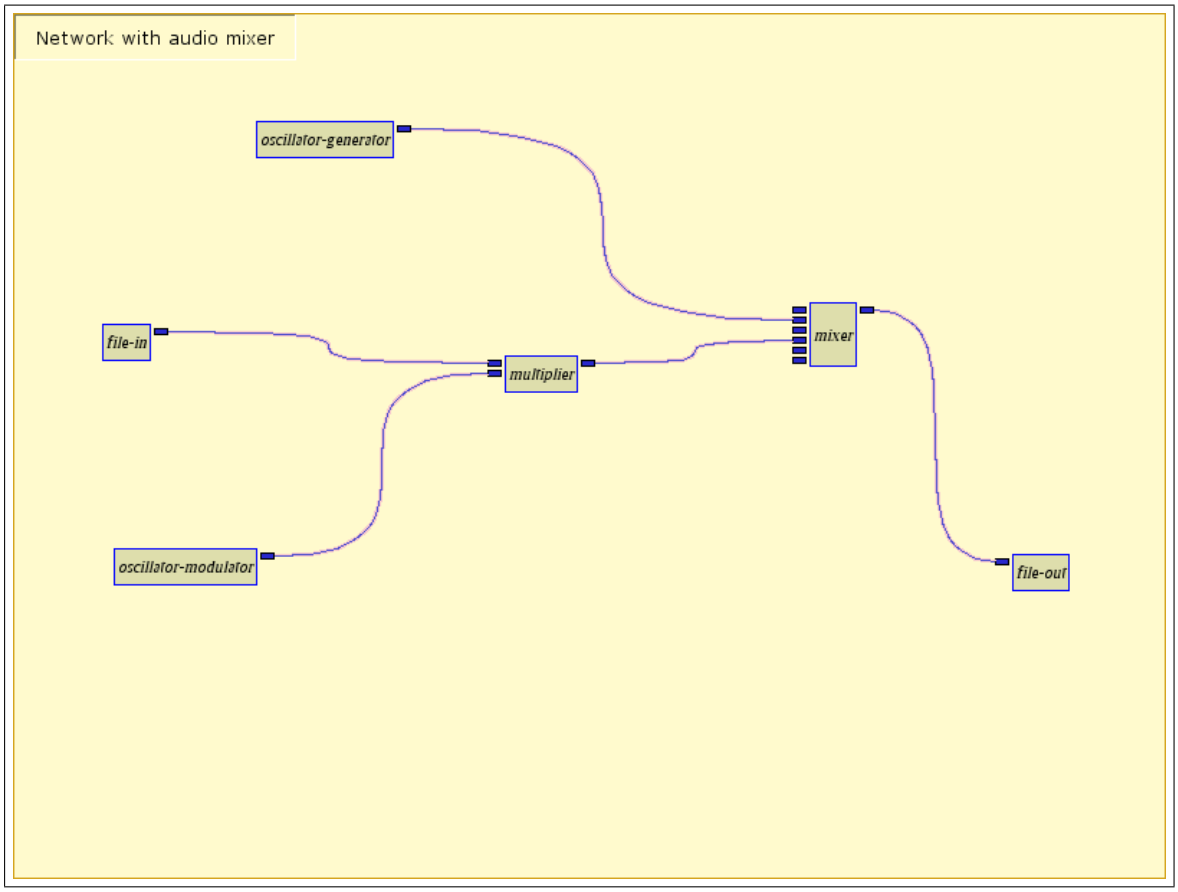

Figura 6.16: Captura de pantalla de l'editor de xarxes

Capítol 7

Conclusions

Un cop acabat ja el disseny i la implementació del sistema, val la pena avaluar tot el que s'ha fet en aquest projecte, les tasques que s'han deixat sense fer i els punts conflictius que s'hauran de resoldre en un futur. Així mateix, faré una avaluació personal del qué ha significat per a mi aquest projecte final de carrera.

7.1 Pensaments sobre la feina feta

Primer de tot, crec que he de dir que encara falta molta feina per fer, tant per la part del mode supervisat de CLAM com per reacTable*. Tot i això, penso que realment s'ha creat una arquitectura que pot aconseguir tots els requeriments que he anat comentant al llarg d'aquest document. En concret, tant la xarxa de processament com el disseny dels nodes com a element automàtic de gestió de les dades permeten compaginar una alta eficiència amb una gran flexibilitat, ja que funciona amb tot tipus de dades, amb el que la transparència esta garantitzada.

Cal destacar que l'ús de les metodologies de desenvolupament iteratiu, especialment les relacionades amb tests, han fet que el codi creat a partir del disseny exposat sigui de bastanta qualitat, especialment en la quantitat de bugs que hi podem trobar.

Si no hagués fet servir aquest mètode probablement el codi tindria en els seus elements més complexos (bàsicament els nodes) bastants errors difícils de detectar.

Així mateix, el disseny de la interfície gràfica fent servir el mòdul de visualització de CLAM, compleix els objectius de baix acoblament, ja que ha estat senzill crear una interfície gràfica d'exemple amb la llibreria gràfica QT, i penso (i espero) que serà igual de fàcil crear-la per l'equip de reacTable* que s'ocupi d'aquest element del sistema.

Així doncs, s'han acomplert en la seva gran majoria els requeriments donats, que al cap i a la fi és el que busca qualsevol projecte informàtic. Això no vol dir que s'hagi implementat tot, sinó que l'arquitectura creada és suficienment complerta com per a fer-se servir per les tasques per les quals ha estat pensada. Personalment crec que això és bastant més important que no pas crear una aplicació concreta, ja que CLAM és un framework d'ajuda al desenvolupament, i no tan sols un programa.

7.2 Línies futures de treball

Com ja he dit, falta molta feina per fer. El mode supervisat està en ple desenvolupament, i per tant, tot basant-se en el disseny presentat, de ben segur que es canviaran algunes coses tot intentant ajustar-se encara més als objectius marcats.

A més, tot i que la serialització de la xarxa de processament està ja dissenyada, no m'ha donat temps a implementar-la. Quan aquesta estigui acabada, tindrem una interfície complerta de la xarxa, de tal manera que podrem editar-la des de XML, tot passivant i activant els objectes, o també modificar-la com es pot fer actualment, a través de codi o amb la interfície gràfica.

Per la part de la interfície gràfica, queden algunes coses per afegir, com pot ser el suport a la serialització, o intentar millorar-ne l'efiència. Per suposat, també caldrà desenvolupar la interfície gràfica de reacTable*. Gràcies al mòdul de visualització i al disseny proposat, no caldrà partir de zero per a fer això.

7.3 Avaluacions personals

En tot aquest temps que he estat treballant en aquest projecte final de carrera pensava que no arribaria mai el moment en què, un cop acabat, podria estar pensant qué és el que m'ha marcat fer-lo. Suposo que és el fet més important en una enginyeria, i realment mentre l'he estat fent ho he anat veient, ja que és quan realment t'enfrontes personalment a una tasca complexa, avaluant totes les alternatives i opcions que et van passant pel cap.

Suposo que això és el que es busca en un enginyer, més que no pas habilitat en el codi o facilitat en l'estudi; una gran capacitat per a avaluar els avantatges i desavantatges de cada solució, per tal de trobar una resposta adequada als problemes que se'ns aniran plantejant. No sé si realment he aconseguit triar les millors opcions possibles en tot moment, però penso que el sol fet d'intentar-lo ja ha valgut la pena.

En un altre nivell, m'ha agradat molt el tema del meu projecte final de carrera, m'entusiasma la música i tot el que l'envolta, i poder treballar en tasques que hi estan estretament relacionades és per mi un privilegi. A més també he pogut madurar molt com a enginyer, en el sentit que el disseny d'una arquitectura com la del mode supervisat no és trivial, i he aplicat tècniques i metodologies que farà tan sols un any ni sabia que podien existir.

Per tot plegat, estic content del projecte, dels seus resultats (tot i que amb matisos), i del que m'ha reportat com a culminació dels estudis.

Capítol 8

Agraïments

Bibliografia

[1] AGNULA.
A gnu/linux audio distribution.
http://www.agnula.org/.

[2] Arumí, P.
Towards a flow model for the mtg-classes (draft), 2001.
Internal document of CLAM development team.

[3] Arumí, P.
Clam nodes design, 2002.
Internal document of CLAM development team.

[4] Arumí, P.
Estructures de dades i control de fluxe per clam: A c++ library for audio and music.
Projecte de Final de Carrera d'Enginyeria Superior en Informàtica a la Facultat d'Informàtica de Catalunya (2002).

[5] Beck, K.
Test-Driven development by example, first ed.
Addison-Wesley, 2003.

[6] Bergamaschi, R., Raje, S., Nair, I., and Trevillyan, L.
Control-flow versus data-flow-based scheduling: Combining both approaches in an adaptive scheduling system.
IEEE (1997).

[7] Budd, T.
Introducción a la programación orientada a objetos, primera ed.
Addison Wesley, 1994.

[8] Chaudhary, A., and Freed, A.
Open sound world.
http://cnmat.cnmat.berkeley.edu/OSW/http://cnmat.cnmat.berkeley.edu/OSW/.

[9] Cobleigh, J., Clarke, L., and Osterweil, L.
The right algorithm at the right time: Comparing data flow analysis algorithms for finite state verification.

Proceedings of the 23rd International Conference on Software Engineering (2001).

[10] CREATE.
Siren.
http://www.create.ucsb.edu/Siren/http://www.create.ucsb.edu/Siren/.

[11] Dannenberg, R., and Brandt, E.
A flexible real-time software synthesis system.
Proceedings of the 1996 International Computer Music Conference (1996).

[12] Davis, P.
Ardour.
http://ardour.sourceforge.net/.

[13] Developers, L. A.
Ladspa.
http://www.ladspa.org/.

[14] development team, C.
Clam: C++ library for audio and music.
http://www.iua.upf.es/mtg/clam/.

[15] Feathers, M.
Cppunit.
http://cppunit.sourceforge.net/.

[16] Foundation, F. S.
Free documentation license.
http://www.fsf.org/licenses/fdl.txt.

[17] Foundation, F. S.
General public license.
http://www.gnu.org/copyleft/gpl.html.

[18] Fowler, M.
Refactoring. Improving the design of existing code, first ed.
Addison Wesley, 2000.

[19] Freed, A., and Chaudhary, A.
Music programming with the new features of standard c++.
Proceedings of the 1998 International Computer Music Conference (1998).

[20] Freed, A., Chaudhary, A., and Wright, M.
An open architecture for real-time music software.
Proceedings of the 2000 International Computer Music Conference (2000).

[21] Gamma, E., Helm, R., Johnson, R., and Vlissides, J.
Design Patterns-Elements of Reusable Object-Oriented Software.
Addison-Wesley, 1995.

[22] García, D.

Suport de xml/mpeg-7 per una llibreria de processat d'àudio i música.
Projecte de Final de Carrera d'Enginyeria en Informàtica a la Salle (2002).

[23] García, D., and Amatriain, X.
Xml as a means of control for audio processing, synthesis and analysis.
Proceedings of MOSART Workshop on Current Research Directions in Computer Music. Barcelona (2001).

[24] IRCAM.
jmax: a visual programming environment.
http://www.ircam.fr/equipes/temps-reel/jmax/en/index.php3http://www.ircam.fr/equipes/temps-reel/jmax/en/index.php3.

[25] Jeffries, R.
What is extreme programming?
http://www.xprogramming.com/xpmag/whatisxp.htm.

[26] Jordà, S.
Faust music on line.
http://www.iua.upf.es/ sergi/FMOL/.

[27] Jordà, S.
Sonigraphical instruments: From fmol to the reactable*.
NIME03, Montreal, Maig 2003 (2003).

[28] Karvelas, M. S. . P.
Designing for schedulability: Integrating schedulability analysis with object-oriented design.
12th Euromicro Conference on Real-Time Systems (Euromicro-RTS 2000) (2000).

[29] Lamport, L.
LaTeX, A document preparation system, second ed.
Addison Wesley, 1994.

[30] Lerr, J. B. . E. A.
The token flow model.
Data Flow Workshop, Hamilton Island, Australia (1992).

[31] Library, F. L. T.
Fltk library.
http://www.fltk.org/http://www.fltk.org/.

[32] Loy, G.
The carl system: Premises, history and fate.
Computer Music Journal (2002).

[33] McCartney, J.
Rethinking the computer music language: Supercollider.
Computer Music Journal (2002).

[34] Meyers, S.
Effective C++ 50 specific ways to improve your programs and designs, second ed.
Addison Wesley, 1998.

[35] MTG.
Music technology group, iua.
http://www.iua.upf.es/mtg/.

[36] Naumovich, G.
Using the observer design pattern for implementation of data flow analyses.
Polytechnic University (2002).

[37] Puckette, M.
Pure-data.
http://www.pure-data.org/http://www.pure-data.org/.

[38] Puckette, M.
Pure data: another integrated computer music environment.
Proceedings, Second Intercollege Computer Music Concerts, Tachikawa, Japan. (1997).

[39] Puckette, M.
Max at seventeen.
Computer Music Journal (2002).

[40] Ramírez, M.
Clam visualization module: Disseny d'un mòdul de visualització o-o per a un entorn de processament d'Àudio multiplataforma.
Projecte de Final de Carrera d'Enginyeria Tècnica en Informàtica a la Universitat Pompeu Fabra (2002).

[41] Robledo, E.
More rumiations on nodes and flow control.
Internal document of CLAM development team.

[42] Robledo, E.
Some supervised mode experimental code, 2002.
Internal document of CLAM development team.

[43] Scaletti, C.
Computer music languages, kyma and the future.
Computer Music Journal (2002).

[44] Scott, M. F. . K.
UML Gota a Gota, 1st ed.
Addison Wesley, 1997.

[45] Sommerville, I.
Ingeniería de Software, sexta ed.
Addison Wesley, 2002.

[46] Steinberg.
Cubase i plugins vst.
http://www.steinberg.net/en/.

[47] STL.
Standard template library.
http://www.sgi.com/tech/stl/.

[48] Stroustrup, B.
The C++ Programming Language, third ed.
Addison Wesley, 1997.

[49] Stroustrup, B.
An overview of the c++ programming language.

[50] Trolltech.
Qt library.
http://www.trolltech.com/products/qt/index.html.

[51] Zicarelli, D.
How i learned to love a program that does nothing.
Computer Music Journal (2002).

Índex alfabètic

Abstracció, 39
AGNULA, 65
ANSI, 40, 41
Ardour, 52
Asincronisme, 58

Bjarne Stroustrup, 40
Buffer, 22

C, 38, 66
C Sharp, 38
C++, 38, 44, 45, 66
C++ Library for Audio and Music, 21
callbacks, 60
CLAM, 13, 21, 42, 44, 51, 65, 76, 93
Classes, 39
Compilador, 41, 46
Concurrència, 40
Control, 22
Control de fluxe, 21
ControlFlow, 22
CPPUnit, 48
CVS, 48

DataFlow, 22
Domini espectral, 22
Domini temporal, 22
Dynamic Types, 66

Eficiència, 39, 41, 43
Emacs, 47
eXtreme Programming, 74

Flexibilitat, 41
FlowControl, 21
FLTK, 42, 65
Fluxe de control, 22
Fluxe de dades, 22
FMOL, 17, 55
framework, 65

g++, 46
Garbage collector, 42
GCC, 41
gcc, 46
GNU, 41, 46
GNU/Linux, 41, 47, 52, 65
GPL, 44, 45, 52, 66

Herència, 39

ISO, 40, 41

Jack, 52
Java, 38
jMax, 44, 55

LADSPA, 55
Lisp, 38
Llenguatges de programació, 40
Llenguatges interpretats, 43

Mètodes, 39
MAX, 54, 61, 63
MAX/MSP, 55
MFC, 45
MIDI, 21, 52, 61
Mode no Supervisat, 14
Mode no supervisat, 68
Mode no Supervisat de CLAM, 22
Mode Supervisat, 15
Mode supervisat, 68, 74

Mode Supervisat de CLAM, 22
MTG, 44
Multiplataforma, 65
MVC, 93

N-Music, 53
Network, 22
Node, 22
Nodes, 69, 77

Objecte de processament, 21
Objectes, 39
Objectes de processament, 66
Objective C, 44
Ordre d'execució, 22
Orientació a objectes, 38
OSW, 44

Paradigma funcional, 38
Paradigma lògic, 38
Paradigma procedural, 38
Paradigmes de Programació, 37
Pascal, 38
PD, 44, 55
Planificador, 22
polling, 60
Port, 22
Processing Data, 67
Processing Object, 21
Processing Objects, 66
Prolog, 38

QT, 42, 45, 65, 97

ReacTable*, 17
reacTable*, 21, 50, 51, 62, 74
Rebirth, 53
Richard Stallman, 47
RTSKED, 54

Scheduler, 22
SDIF, 65
Serialització, 89
signal, 60
Simula, 40
Sincronisme, 58
Siren, 44
slot, 60
Smalltalk, 38, 44
Standard Template Library, 42
STL, 42

Template, 42
Templates, 66
Temps real, 52
Test-Driven Development, 75
Test-Driven development, 48
The ReacTable*, 17
The reacTable*, 21
Trolltech, 45

Umbrello, 48
UML, 48

VST, 55

Xarxa, 22
Xarxa de processament, 70, 87
XML, 65, 89