Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria
Secció d'Enginyeria en Informàtica

SaxSMS + ScoreMan

Tècniques de segmentació espectrals per l'anàlisi i síntesi de sons monofònics en SMS

Memòria del projecte fi de carrera d'Enginyeria en Informàtica presentada per en Oscar Mayor Soto i dirigida per en Josep Lluís Arcos i en Carles Sierra de l'Institut d'Investigació en Intel· ligència Artificial del CSIC.

Bellaterra, Febrer del 2001

El sotasignant, Sr. Carles Sierra i Garcia, de la Universitat Autònoma de Barcelona i de l'Institut d'investigació en Intel· ligència Artificial del CSIC

Certifica:

Que el treball corresponent a la present memòria ha estat realitzat sota la seva direcció per en Oscar Mayor Soto

Bellaterra, 14 de Febrer de 2001

Signat: Carles Sierra i Garcia

A tots aquells que m'han fet costat durant
la realització d'aquest projecte...

Gràcies

Agraïments

Voldria donar les gràcies a moltes persones, així que possiblement em deixaré a algú, espero que no ho tingueu en compte i m'ho sabeu perdonar.

En primer lloc vull donar les gràcies a la meva família i en especial als meus pares Vicente i Elvira i al meu germà Vicenç que sempre m'han fet costat als moments més difícils.

També vull donar un fort agraïment a tots els meus companys de carrera, els meus amics de fora de la universitat, companys de grup musical i els meus companys de feina del Music Tecnology Group de l'Institut de l'Audiovisual de la UPF, sense tots ells la meva vida seria molt avorrida.

Com no, he d'agrair també a tots els meus professors de carrera i en especial als meus professors de saxo, Gorka Benítez i Jordi Santanah.

Finalment, i no per això menys important vull agrair a en Ramón López de Mántaras, Josep lluís Arcos, Jordi Bonada i Carles Sierra els ajuts i suport que m'han donat durant la realització d'aquest projecte.

Moltes gràcies a tots.

Universitat Autònoma de Barcelona
Secció d'Enginyeria en Informàtica

Unitat de Processament d'Imatges i Intel· ligència Artificial

Projecte de fi de carrera

SaxSMS + ScoreMan

Tècniques de segmentació espectrals per l'anàlisi i síntesi de sons monofònics en SMS

Oscar Mayor Soto

Bellaterra, Febrer del 2001

Projecte desenvolupat a l'IIIA (CSIC) sota la supervisió
del Dr. Josep Lluís Arcos i en Carles Sierra

Índex

Introducció **1**

Objectius i mòduls del projecte **3**

I TREBALL EXISTENT **6**

Capítol 1: SMS **7**

1.1- Què és SMS? 7

1.1.1- Anàlisi 8

1.1.2- Síntesi 10

Capítol 2: SaxEx **12**

2.1- Què és Saxex? 12
2.1.1- Fase de raonament (CBR) 14

II TREBALL REALITZAT **17**

Capítol 3: Millores i Aportacions a SMS **18**

3.1- Paràmetres d'anàlisi a SMS 18
3.2- Millores en la detecció de pitch 21
3.2.1- Detecció de pics i fonamental 21
3.2.2- L'algorisme TWM 23
3.2.2.1- Millores sobre TWM 24
3.2.3- Reanàlisi dels atacs 26
3.3- Segmentació en regions 28
3.3.1- Variacions de l'amplitud 29
3.3.2- Variacions de la freqüència 31
3.3.3- Tipus d'articulacions 33
3.3.4- Algorisme de detecció de regions 35
3.3.5- Representació de les regions 42

Capítol 4: ScoreMan **46**

4.1- Què és ScoreMan? 46
4.2- Fitxers de ScoreMan 47
4.3- Representació interna i jerarquia de classes 50
4.4- Funcionament 55

4.4.1- Canvis d'amplitud 56
4.4.2- Canvis de durada (time stretching) 58
4.5- El GUI 61
4.5.1- Visualització de fitxers SaveRegions 62
4.5.2- Visualització de fitxers de canvis SaxEx 63
4.5.3- Execució de fitxers Script 65
4.6- El model Client-Servidor 67

Capítol 5: Proves 71

5.1- Els casos no expressius 72
5.2- Els casos expressius 75
5.3- Conclusions de les proves realitzades 80
5.4- Partitures 81

Capítol 6: Conclusions 84

Referències 86

Introducció

La música, el processament del senyal i la informàtica són tres camps d'estudi aparentment molt separats però que realment estan molt relacionats, podríem pensar que la música està molt lluny de ser considerada com a estudi científic, però ja des del temps de Pitàgores (ss. VI-V a. C.) es van trobar unes lleis que regien els sons, i per extensió, la música. Pitàgores ho va plantejar explicant la relació que hi havia entre la longitud d'una corda i els intervals harmònics més senzills, d'aquesta manera va dir que si al polsar una corda es produeix un so corresponent a una nota, al reduir la distància de la corda a la meitat, el so produït serà la mateixa nota però una octava superior, és a dir la freqüència de vibració de la corda serà el doble. Amb aquest procediment, trobà les relacions, expressables matemàticament (científicament) entre octaves, quintes i quartes [1][2]. Més tard Aristòtil (384-322 a. C.) va fer nous descobriments com per exemple que freqüència i velocitat de propagació eren conceptes diferents i que els sons estaven compostos per una sèrie d'harmònics. Arran d'aquests descobriments, molts científics comencen a investigar aplicant aquestes lleis a determinats instruments musicals, fins que al segle XVI, Marsenne arriba a la conclusió de que un so està format per un so natural i un conjunt d'altres sons, amb aquest descobriment introdueix el concepte de

timbre i proposa l'espectre com a mètode per diferenciar timbres, que es el que diferencia una mateixa nota produïda per dos instruments diferents. Ja al segle XIX, els experiments s'intensifiquen i científics com Hemholtz, Rayleigh o Koenig estableixen els fonaments bàsics de l'acústica moderna. Actualment, l'aparició de computadores cada cop més potents i l'evolució de la informàtica en general al segle XX, està permetent que es doni molta importància al processament digital del senyal, per analitzar i modelar els sons que produeixen els intruments musicals actuals, o bé per crear-ne de nous amb procediments de síntesi digital.

Objectius i Móduls del Projecte

Aquest projecte s'engloba dintre d'altre projecte més gran anomenat SaxEx [4][5]. L'objectiu bàsic de SaxEx és poder treballar amb un sistema que afegeixi expressivitat a una interpretació no expressiva d'una melodia produïda per un instrument monofònic de vent, com podria ser un saxo (tenor, alto, soprà) o un clarinet. Aquesta melodia pot ser interpretada per un instrument real, o bé un sintetitzador digital, i el que es vol aconseguir es dotar al so resultant d'una expressivitat similar a la que s'aconsegueix en les interpretacions fetes per humans, així com poder generar diferents tipus d'expressivitat com podria ser tristesa, alegria, tendresa, agressivitat,... El que caracteritza a un instrument monofònic i el diferencia d'un polifònic és el fet de només poder fer sonar una nota al mateix temps, és a dir no poden produir acords, tot i que hi ha hagut alguns intèrprets que amb un saxo, utilitzant avançades tècniques d'expulsió de l'aire i posicions complexes de les claus aconsegueixen fer sonar dues notes diferents al uníson. El fet de treballar amb instruments de vent de la família de la fusta (reed instruments) i no amb qualsevol instrument en general és perquè cada instrument té un timbre característic i és molt difícil poder fer un sistema que funcioni independentment de l'instrument que interpreta la melodia.

SaxEx, a la vegada fa servir una altra eina anomenada *SMS* [3], encarregada d'analitzar i sintetitzar sons musicals, la finalitat que es vol obtenir amb el projecte que ocupa aquesta memòria és poder fer el procés d'afegir expressivitat d'una manera automàtica, sense haver-se de preocupar de com sona la cançó, o a quin segon de la cançó hem de fer un determinat canvi. Fins ara es presentava un problema, i era que els resultats de l'anàlisi *SMS* no eren els desitjats al treballar amb instruments de vent i hi havia una manca de contingut musical, necessària per *SaxEx* a l'hora de fer el procés "d'expressivització". Llavors aquest projecte tracta de fer millores i afegir funcionalitat al software *SMS* així com fer una connexió entre aquests dos projectes (*SaxEx* i *SMS*) per tal d'integrar-los com un de sol.

Així doncs, a l'inici del projecte es disposava de dues eines, que són:

- *SMS*, que és un software d'anàlisi i síntesi d'àudio, basat en representació mitjançant sinusoides mes soroll [3], aquest software ens servirà per analitzar els sons i tenir una representació espectral de la melodia, i a partir d'aquesta representació, sintetitzar el nou so amb l'expressivitat desitjada. *SMS* s'explica amb més deteniment en un capítol posterior.

- *SaxEx*, que és un software capaç de generar interpretacions musicals expressives, basat en tècniques de raonament basat en casos (CBR), que amb la possibilitat d'interacció per part de l'usuari, permet generar diferents tipus d'expressivitat basades en paràmetres d'afectivitat com podrien ser: alegria, agressivitat o dinamisme [4][5]. SaxEx també s'explica en detall en un capítol posterior.

El problema que apareix és que *SMS* és un sistema més aviat orientat a anàlisi i síntesi de veu i al treballar amb sons provinents d'un saxo no produeix una anàlisi molt acurada, i a més a més, els paràmetres amb els quals treballa són de massa baix nivell, amb el conseqüent problema de no poder treballar a nivell musical, sinó a nivell de freqüències o de temps. L'objectiu preliminar és el de la millora de l'anàlisi, és a dir, aconseguir que *SMS* produeixi una sortida on tinguem informació de més alt nivell, com podrien ser les notes que hi ha en cada moment o la durada dels atacs, o el tipus d'articulació de les notes (legato, stacato), o qualsevol altre informació útil per facilitar el procés "d'expressivització" que realitza *SaxEx*. Un cop s'obté una sortida del *SaxEx* amb la informació expressiva, que serà també d'alt nivell (musical), s'haurà de convertir a un format de fitxer que entengui *SMS*, per tal de sintetitzar el nou so, i poder sentir la nova interpretació.

A continuació podem veure un diagrama amb els blocs del projecte. Com podem veure al dibuix, les parts en groc representen els mòduls que o bé han sigut ampliats o millorats o bé s'han fet de nous per completar el projecte inicial:

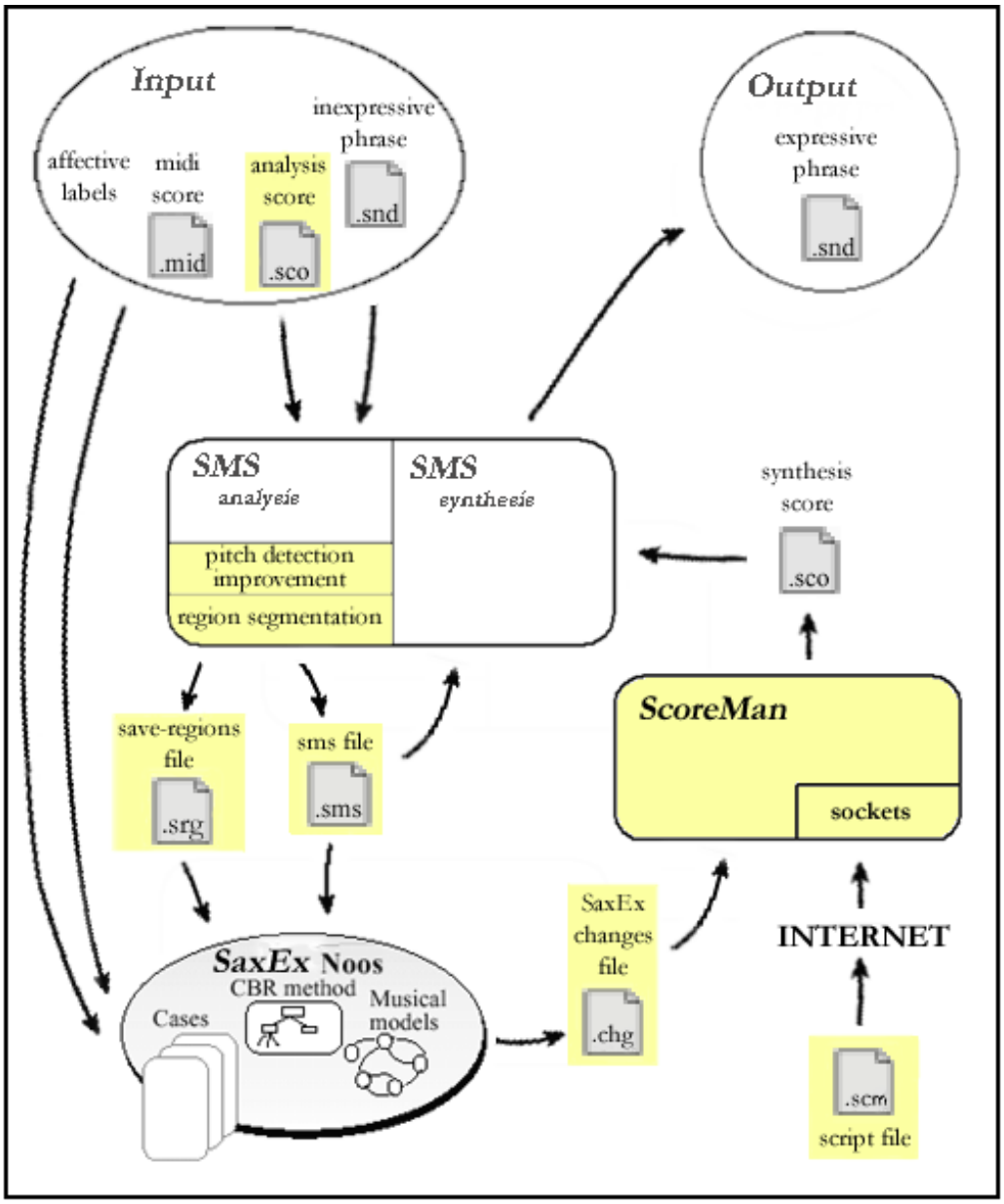


Figura 0: Diagrama de blocs del projecte.

TREBALL EXISTENT

Capítol 1: SMS

1.1- Què és SMS?

SMS (*Spectral Modeling Síntesis*) és un sistema desenvolupat originàriament per Xavier Serra i Julius Smith a la Stanford University que està en constant desenvolupament pels membres del MTG (Music Tecnology Group) de la Universitat Pompeu Fabra. És un sistema d'anàlisi i síntesi de so basada en models espectrals, que permet, mitjançant procediments d'anàlisi, extreure un conjunt de paràmetres del so original que ens aportin una gran flexibilitat envers possibles transformacions o extracció d'informació del so original.

La finalitat principal de SMS és, mitjançant l'anàlisi, extreure una representació musical del so per així poder manipular uns paràmetres musicals mantenint una alta qualitat de so i poder sintetitzar nous sons, o bé fer processos de reconeixement de cançons, veu, separar un instrument solista dins d'una secció, etc.

Entre les possibles representacions del so que podem fer servir amb models spectrals, com la *Short-Time Fourier Transform (STFT)*, la *Sinusoidal*, la *Sinusoidal plus Residual* o amb *High Level Atributes*, SMS fa servir una combinació de les dues darreres per obtenir un equilibri entre qualitat de so i temps i espai computacional, així modelitza el so original com a Sinusoides i Soroll i s'intenten extraure el màxim nombre d'atributs possibles per tal de mantenir un rendiment òptim i una bona qualitat de so. Els dos subcomponents en que es descomposa el so són: el component periòdic expressat mitjançant sinusoides que representen la part harmònica o determinista del so, és a dir el so que produiria un instrument "ideal" i el component residual, que correspon al soroll, és a dir el so produït per l'articulació de les claus en un instrument de vent, el soroll de la pua en una guitarra o la respiració d'una cantant.

El procés d'anàlisi i síntesi a *SMS* està basat en la idea de finestra en el temps, permetent així una gran flexibilitat a l'hora d'analitzar o sintetitzar un tros determinat dintre de la seqüència total.

Els components d'un so en un temps t, són modelats com:

$$s(t) = \sum_{r=1}^{R} A_r(t) \cdot \cos[\theta_r(t)] + e(t)$$

on $A_r(t)$ i $\theta_r(t)$ representen l'amplitud i fase en el temps t per a la sinusoide r-èssima i $e(t)$ és el component de soroll en el temps t.

1.1.1- Anàlisi

El procés d'anàlisi consta bàsicament de 6 etapes (veure figura 1.1):

- Càlcul de la magnitud i fase de l'espectre del so, mitjançant la Transformada Ràpida de Fourier (FFT)
- Peak detection (detecció de pics)
- Fundamental Pitch detection (detecció de freqüència fonamental)
- Peak continuation (continuació de pics)
- Residual Anàlisis
- Extracció d'atributs d'alt nivell o paràmetres musicals.

La primera fase d'anàlisi està molt íntimament lligada a la teoria de Fourier, primer es convoluciona el so original amb una finestra d'anàlisi que actua com a filtre, suavitzant el so als extrems d'aquesta i s'aplica llavors l'algorisme de la Transformada Ràpida de Fourier (FFT), que ens dona com a resultat l'espectre en magnitud i fase del so original.

Una segona fase consisteix en buscar al domini freqüencial (com a resultat de la fase anterior), la freqüència dels pics que compleixen una determinada amplitud mínima (entre tots els màxims locals), i els seus harmònics (2 * f, f / 2, ...) per tal de construir una llista de candidats a freqüència fonamental del so (suposant que aquest sigui harmònic).

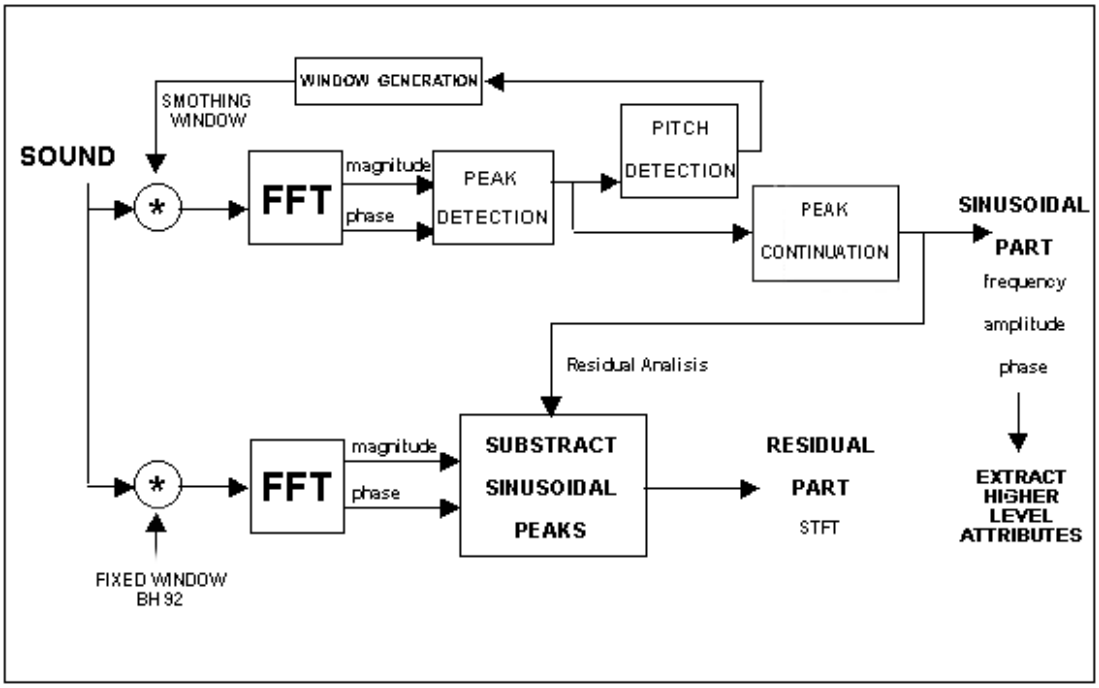


Figura 1.1: Procés d'anàlisi a SMS.

Un cop tenim els candidats a *pitch*, mitjançant l'algorisme *Two-Way Mismatch* (TWM) es decideix quin dels pics candidats es més apropiat com a fonamental, es tracta de buscar el divisor comú de les sèries harmòniques dels candidats.

La continuació de pics fa referència al càlcul de les trajectòries dels parcials en el temps, un cop tenim decidit el *pitch* d'una zona ja analitzada, és molt útil fer servir

aquesta informació per canviar la mida de la finestra d'anàlisi a mesura que es va analitzant el so i així aconseguir un procés d'anàlisi més acurat.

Per obtenir el component residual del so, el que es fa és generar mitjançant síntesis aditiva el component sinusoidal, prèviament calculat, i aquest component s'extrau del so original en el domini temporal, quedant només el soroll que no forma part dels harmònics del so. Llavors aquest soroll es guarda com a *Short Time Fourier Transform*. D'aquesta manera, les transformacions es fan sobre la part sinusoidal, i després s'afegeix la part residual, ja que a moltes transformacions no és adequat fer canvis sobre el soroll.

L'últim pas del procediment d'anàlisi que encara s'està desenvolupant i que es un dels objectius d'aquest projecte, consistiria en extraure atributs d'alt nivell o paràmetres musicals, que ens ajudin a fer una bona síntesi del so original controlant paràmetres musicals i sense haver-se de preocupar d'atributs de massa baix nivell sense cap sentit per a un compositor o usuari final no expert en temes de processament del senyal. Aquesta fase te més sentit en el cas d'estudi d'un instrument concret, ja que el procediment variarà bastant d'un instrument a un altre. Possibles atributs útils serien les regions d'una nota (*attack*, *steady state*, *release*), freqüència mitja d'una regió, tipus de transició entre les notes (*stacatto*, *legatto*) o extracció de *vibrato*.

1.1.2- Síntesi

El procés de síntesi en SMS es fa totalment al domini freqüencial i les transformacions es defineixen majoritàriament com a funcions envolupants en el domini temporal (veure figura 1.2). En el cas de no fer cap tipus de transformació sobre el so analitzat, el procés de síntesi es fa totalment sense pèrdua de qualitat (*loseless*), és a dir el so resultant és idèntic a l'original, ja que es fa la Transformada Ràpida Inversa de Fourier (IFFT) de la part sinusoidal i residual i s'afegeixen totes dues parts per formar el so sintetitzat. En cas d'aplicar transformacions, la qualitat del so sintetitzat, també és molt bona i natural, ja que les transformacions es solen aplicar independentment sobre la part sinusoidal i residual.

Les transformacions principals que es poden aplicar sobre un so prèviament analitzat amb SMS es poden agrupar en quatre tipus:

- Transformacions d'amplitud.
- Transformacions de freqüència.
- *Time-stretching* (estirament del temps).
- Modulacions de freqüència o amplitud.

Així, suposant que hem fet l'extracció d'atributs d'alt nivell, si tenim les notes que hi ha al so prèviament analitzat, descomposades en el seu respectiu atac, regió estable i final, podem fer canvis en el volum d'una nota, o canviar la freqüència o durada de la nota mitjançant *time-stretching*, fent-ho sobre la part de la nota adient, majoritàriament sobre la part estable de la nota, per evitar sons indesitjables en la síntesi com podria ser, per exemple, un atac massa llarg, en el cas de fer un estirament de la nota.

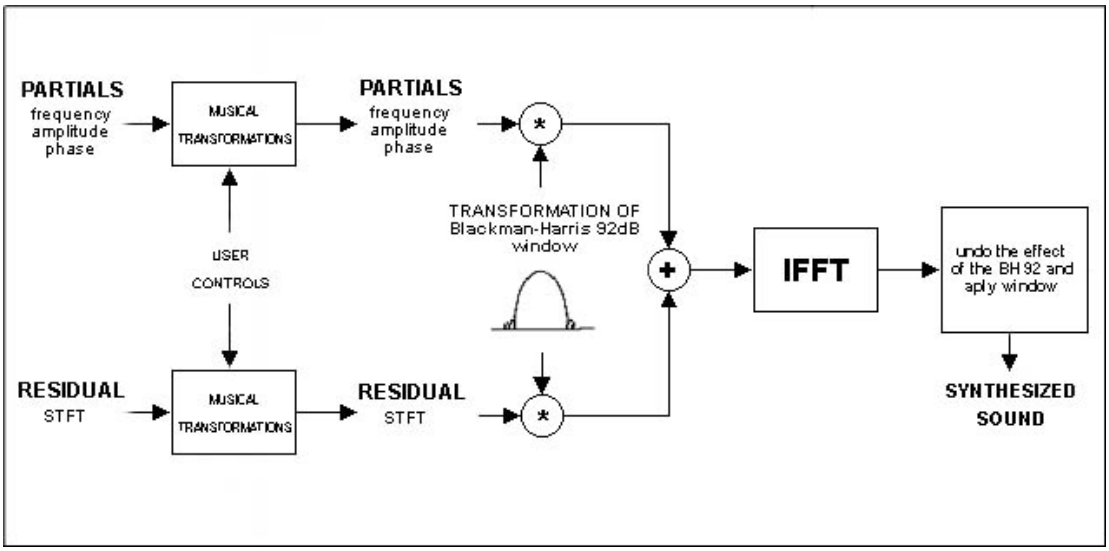


Figura 1.2: Procés de síntesi a SMS.

Capítol 2: SaxEx

2.1- Què és SaxEx?

SaxEx és un sistema realitzat per Josep Lluís Arcos i Ramón López de Mántaras al IIIA (Institut d'Investigació en Intel· ligència Artificial) capaç de generar interpretacions musicals expressives, utilitzant tècniques de raonament basat en casos (CBR).

SaxEx sorgeix de la necessitat de fer servir no només els coneixements musicals per tal de generar música, sinó donar un toc humà de la mateixa manera que ho fem les persones, amb un procés d'observació, imitació i experimentació. Per això, el sistema *SaxEx*, utilitza una base de dades amb exemples d'interpretacions expressives reals, per tal de generar-ne de noves amb procediments de raonament basat en casos.

SaxEx ha estat implementat en llenguatge *Noos*, un llenguatge orientat a objecte dissenyat per representar el coneixement en problemes d'aprenentatge i solució (*problem solving and learning*).

Amb la finalitat d'aconseguir uns canvis expressius agradables musicalment, SaxEx fa servir dues teories generals sobre percepció i comprensió musical i coneixements sobre teoria del Jazz, ja que l'entrada del sistema són temes estàndards de Jazz.

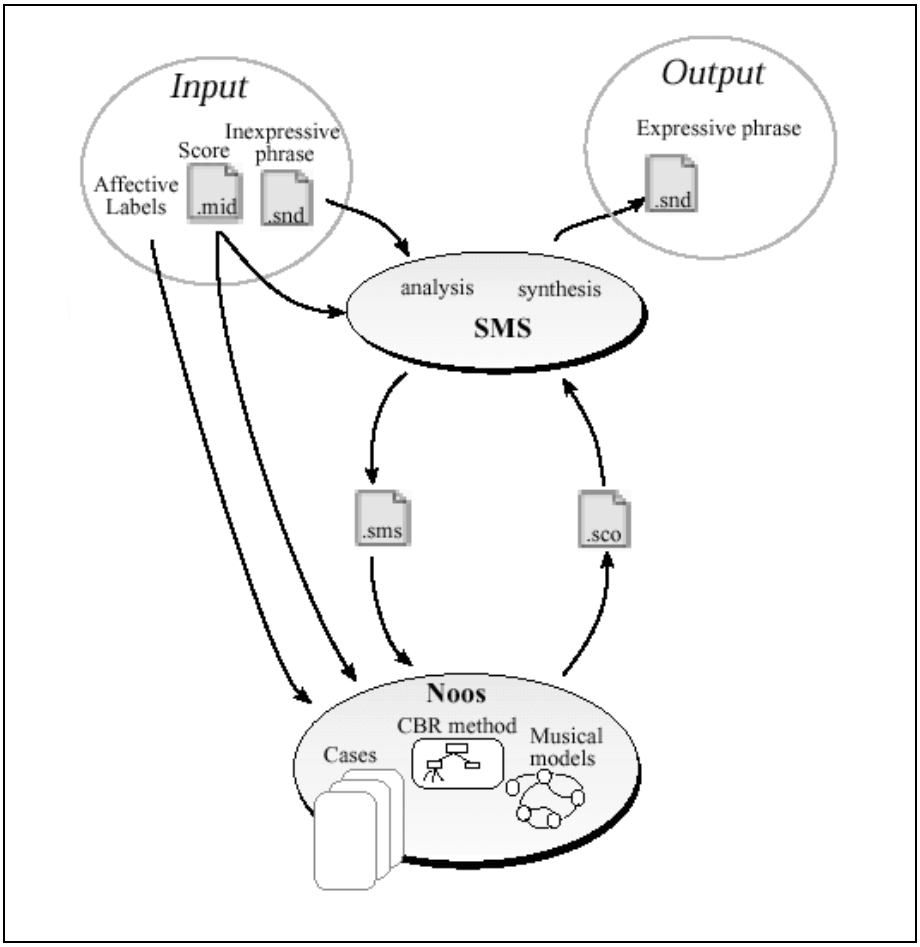


Figura 2.1: Diagrama de blocs de SaxEx (extret de [5]).

L'entrada del sistema *SaxEx* és habitualment (veure figura 2.1):

- Una frase musical descrita per la seva partitura (*score*), com un fitxer MIDI amb informació melòdica i harmònica.

- Un so gravat (la frase musical interpretada sense expressivitat per un humà).

- Valors pels paràmetres afectius utilitzats (tendre - agressiu, trist - alegre, calma - moviment)

La sortida del sistema és (veure figura 2.1):

- Un conjunt de nous fitxers de so obtinguts mitjançant transformacions de l'original, afegint l'expressivitat definida als paràmetres afectius escollits per l'usuari.

El procés de *SaxEx* consta de 3 fases:

- La fase d'anàlisi (SMS).
- La fase de raonament (CBR).
- La fase de síntesi (SMS).

Les fases d'anàlisi i síntesi es fan mitjançant les tècniques de *SMS* explicades al capítol anterior, i la fase de raonament està implementada en *Noos*, i és la fase més important del sistema.

2.1.1- Fase de Raonament (CBR)

Com ja s'ha dit, la fase de raonament és la fase més important, ja que és la fase que realment troba la solució als problemes plantejats (frases musicals on es vol afegir expressivitat). *SaxEx* té una representació interna dividida en dos models: El model del domini (*domain model*) i el model del solucionador de problemes (*problem-solving model*). El primer model ve representat pel coneixement musical, i el segon pel mètode CBR que ens proporciona una seqüència de transformacions expressives d'una frase musical donada.

El model del domini, integra tot el coneixement musical que facilita al *problem-solver* trobar una solució bona. Aquest model incorpora una teoria general basada en la

percepció musical com és el *Narmour's implication/realization (IR) model* [6], una teoria basada en la comprensió musical com és la *Lerdahl and jackendoff's generative theory of tonal music GTTM* [7], i nocions de coneixement específic sobre teoria del Jazz, ja que de moment el sistema està pensat per melodies d'estàndards de Jazz.

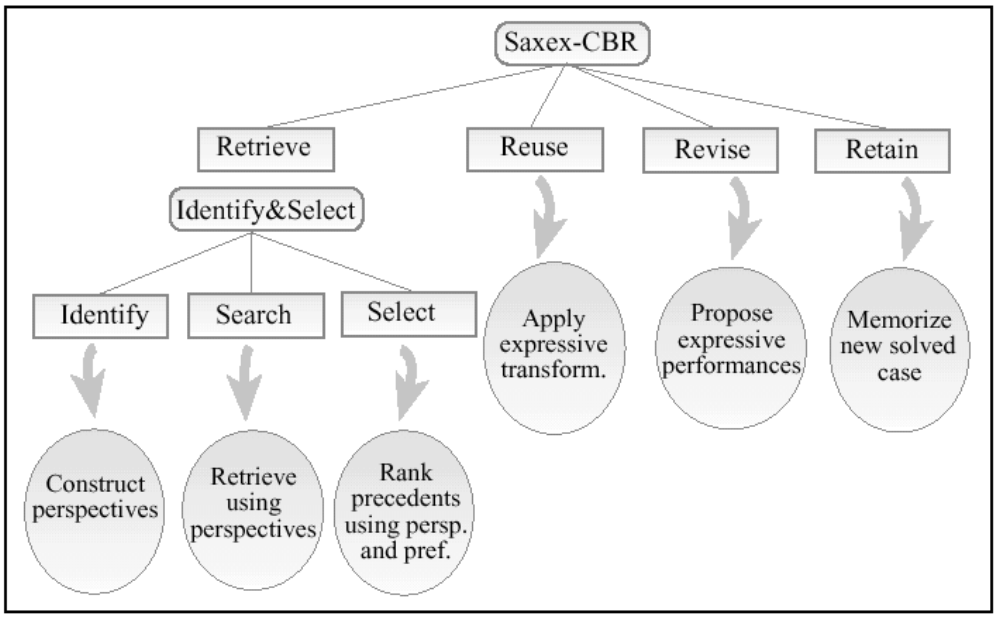


Figura 2.2: Etapes del mètode CBR de SaxEx (extret de [5]).

El model del solucionador inclou el raonador basat en casos (CBR), el procés de raonament consta de 4 etapes (veure figura 2.2):

- *Recuperació*: per cada nota de la melodia, s'escull de la memòria de casos amb les peces interpretades expressivament, el conjunt de notes més semblants al cas del problema. Les notes escollides dependran del tipus d'expressivitat escollida per l'usuari dintre dels paràmetres tendre-agressiu, trist-alegre, calma-moviment). Aquest procés es podria dividir en 3 subprocessos:

    - *Identificació*: fent servir el coneixement musical integrat a *SaxEx*, es determinen les característiques musicalment importants de la nota.

- *Búsqueda*: es fa una cerca a la memòria de casos mitjançant mètodes de recuperació de *Noos* per tal de trobar situacions similars a la nota prèviament analitzada.

- *Selecció*: S'estableix un ordre de preferència dels casos buscats mitjançant els mètodes de preferència de *Noos*, basats en similitud entre la durada de les notes, direcció melòdica o estabilitat harmònica del fragment analitzat.

- *Reutilització*: Un cop es tenen els casos més similars al problema en qüestió, l'objectiu és reaprofitar els canvis que apareixen a la memòria per tal d'aplicar-los al fragment que està sent analitzat, adaptant-los segons convingui.

- *Revisió*: A l'usuari se li presenta un conjunt de solucions perquè pugui escollir entre elles la que més li satisfagui i perquè pugui modificar els paràmetres expressius assignats a una nota específica.

- *Retenció*: Quan s'ha obtingut la solució a un problema nou, en el cas de que l'usuari l'aprovi com a bo, aquesta solució és automàticament integrada a la memòria de casos per tal de poder ser aplicada en futurs problemes.

TREBALL

REALITZAT

Capítol 3: Millores i Aportacions a SMS

3.1- Paràmetres d'anàlisi a SMS

Un cop analitzat i entès el procés realitzat pel software *SMS*, el primer problema que apareix és la manca d'un paràmetre que indiqui el tipus d'instrument que s'està analitzant, molt important pel nostre cas, ja que ens disposem a fer unes millores sobre el programa dependents del tipus d'instrument que s'analitza. D'aquesta manera les modificacions que es faran només afectaran en el cas de que s'estigui analitzant un saxo, en un altre cas el programa es comportarà de la mateixa manera que ho feia abans.

A SMS els paràmetres, tant d'anàlisi com de síntesi, s'especifiquen en un fitxer anomenat *score* (.sco), escrit manualment en format text/ascii que inclou el nom dels paràmetres i els seus valors associats, en el cas de que un paràmetre no figuri en aquest fitxer, al fer l'anàlisi o la síntesi, automàticament se li assigna el seu valor per defecte.

Es va decidir incloure un nou paràmetre envolupant anomenat *InstrumentType* que per defecte pren el valor 0 i pel cas del saxo pren el valor 1. El rang de valors acceptats pel paràmetre és de 0 a 255, és a dir contemplem fins a 256 tipus d'instruments diferents. El fet de que el paràmetre sigui envolupant, vol dir que aquest paràmetre, al igual que molts altres paràmetres de SMS, és una funció que pot variar al llarg del temps, és a dir, podríem tenir un fitxer on a la primera meitat sona un saxo i a la segona meitat sona un piano, llavors si ens interessa analitzar els dos instruments independentment, suposant que 2 sigui el tipus d'instrument piano, el *score* d'anàlisi hauria de contenir una línia de l'estil:

	primera meitat				segona meitat			
InstrumentType	0	1	0.5	1	0.5	2	1	2

La funció es defineix per parelles de punts (*x, f(x)*), d'aquesta manera tenim que des de l'inici fins a la meitat del fitxer (0.5 és la posició mitja del so que s'analitza), l'instrument és el tipus 1 (saxo) i des de la meitat del fitxer fins al final del fitxer (posició 1), el tipus d'instrument és el 2 (piano). Gràficament es pot entendre millor si mirem la figura 3.1:

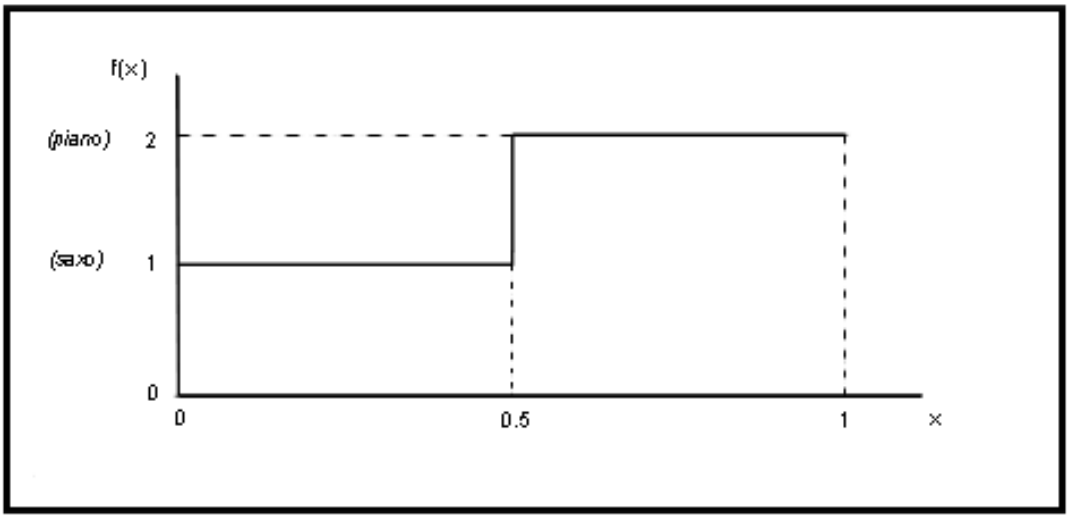


Figura 3.1: Exemple gràfic d'un paràmetre envolupant a SMS.

Per tal de millorar el procés d'anàlisi, s'han ajustat uns paràmetres del fitxer *score* d'anàlisi, referents a la freqüència a la que es buscaran els candidats a freqüència fonamental de les notes que s'analitzen a una melodia, per exemple en el cas d'un saxo alt, el registre avarca des dels 138 Hertz per la nota més greu que correspon a un Db1 fins als 1670Hertz de la nota més aguda que correspon a un Ab3, per tant no tindrà sentit buscar una nota amb freqüència fonamental (*pitch*) fora d'aquests valors. Si tenim informació sobre les melodies que s'interpretaran (com és el nostre cas), podem ajustar els valors d'aquests paràmetres (*LowestPitch* i *HighestPitch*) segons les notes de la melodia que generalment comprenen un registre petit entre els 150 i 800 Hertz, l'ajust d'aquests paràmetres repercuteix en una millor i més ràpida detecció de la freqüència fonamental de cada nota analitzada. Veiem a la figura 3.2 un exemple de fitxer *score* d'anàlisi SMS.

```
InputSoundFile c:\Prog\SMS\test\InWavs\love\lovee11644.wav
OutputSmsFile c:\Prog\SMS\test\OutSMS\love\lovee1A1644.sms
InstrumentType 1
BeginPos 0
EndPos 0.242
FrameRate 345
SineModel 1
ResModel 4
nSines 100
LowestPitch 150
DefaultPitch 300
HighestPitch 800
HighestFreq 22050
AttackReanalysis 1
nAttackReanalysisFrames 15
AttackReanalysisLowPitchMargin 0.95
AttackReanalysisHighPitchMargin 1.05
UseHarmonicHighLowEnergyRatio 0
SaveRegions 1
SaveRegionsFile c:\Prog\SMS\test\srg\love\love_e1A1644.srg
```

Figura 3.2: Exemple de fitxer score d'anàlisi SMS.

Dels altres paràmetres que es veuen al fitxer *score* d'anàlisi de la figura 3.2 ja es parlarà més endavant.

3.2- Millores en la detecció de pitch

Un cop feta l'anàlisi de diversos fragments de melodies interpretades per un saxo, s'observa que la fase d'anàlisi no dóna gaire bons resultats a la detecció de pitch, sobretot a les zones on es produeixen els atacs de les notes, ja que és una zona on es produeix molt soroll i pot influir molt en la no correcta decisió de la freqüència fonamental de la nota que sonarà a partir d'aquest punt. Un altre problema que apareix és que en determinades circumstàncies, la decisió del pitch de la nota és errònia, donant com a resultat la freqüència de l'octava superior. A continuació veurem quin és el procés de detecció de pitch que es fa servir per establir quines millores es poden fer.

3.2.1- Detecció de pics i freqüència fonamental

El primer pas abans de començar a fer la detecció de *pitch* consisteix en convolucionar el so original amb una finestra d'analisi (W_s) de tamany variable que suavitza el so als extrems, aquest procès, al igual que molt del procesament SMS es realitza a nivell de frames, és a dir es tracta el so com si fossin fotogrames i s'analitzen per separat, un cop feta la convolució del so amb la finestra, s'aplica la transformada ràpida de Fourier sobre el resultat per obtenir l'espectre de magnitud i fase del so que s'està analitzant. Un cop tenim l'espectre, hem de determinar amb l'algorisme de *continuació de pics*[1] quins pics es poden considerar parcials i quins soroll i quins pics contribueixen amb més pes (magnitud més gran) a l'espectre, per així formar una llista de candidats a pitch.

La finalitat bàsica de la detecció del pitch en cada moment és el fet de poder

---

1 El procés consisteix en que per a cada trajectòria s'escull el pic de freqüència i magnitud més pròxims. S'ha de tenir present, però que dues trajectòries poden escollir el mateix pic *(peak conflic*t). Aleshores s'ha de triar a quina trajectòria s'ajusta millor el pic, i buscar un altre pic per a l'altra trajectòria. En el cas d'un so harmònic, les trajectòries s'ajusten a unes guies que a cada *frame* es recalculen segons el *pitch* que s'ha trobat i la freqüència del pic que s'ha agafat al *frame* anterior.

representar la part sinusoidal del so, els parcials, fent una assumpció harmònica, d'aquesta manera, passaríem d'una representació com a:

$$s(t) = \sum_{r=1}^{R} A_r(t) \cdot \cos[\theta_r(t)] + e(t)$$

on $A_r(t)$ i $\theta_r(t)$ representen l'amplitud i fase en el temps t per a la sinusoide r-èssima i $e(t)$ és el component de soroll en el temps t, a una representació on agrupem els parcials relacionats harmònicament, i deixem per la part residual la part no harmònica i el soroll del so, així obtenim una representació:

$$s(t) = \sum_{r=1}^{R} A_r(t) \cdot \cos[2\pi r \cdot (f_0(t) + \Delta_r \cdot f_0(t)) \cdot t + \Psi_r(t)] + e(t)$$

on $A_r(t)$ i $\Psi_r(t)$ representen l'amplitud i fase en el temps t per a la sinusoide r-èssima, Δ_r és la desviació de la freqüència de la sinusoide r-èssima, suposant una sèrie harmònica perfecta normalitzada respecte a la freqüència fonamental f_0 i $e(t)$ és el component de soroll en el temps t.

Aquesta nova representació del so és molt més flexible i permet una anàlisi més acurada en instants de temps posteriors, ja que l'amplada de la finestra d'anàlisi es reduirà en el cas de que la freqüència fonamental detectada en l'instant de temps anterior augmenti o s'augmentarà en el cas de que la freqüència fonamental detectada en l'instant de temps anterior disminueixi per així adaptar-se bé a la resolució freqüencial del so i poder detectar els pics amb més facilitat. El problema sorgeix quan el so d'entrada no és harmònic, i per resoldre aquest cas hi ha un mecanisme que quan no es detecta *pitch*, el so es classifica com a *unpitched* i tot l'espectre es guarda com a residual [8].

El procediment utilitzat per fer la detecció de *pitch* consisteix en a partir d'una llista de pics candidats a *pitch* amb la seva respectiva magnitud i freqüència, determinar quina freqüència fonamental és la més correcta en cada moment, calculant la sèrie harmònica que dóna el mínim error al construir-la amb els pics espectrals que tenim a la

llista de candidats, aquest mètode esta basat en l'algorisme Two-Way Mismatch (TWM) proposat per Maher i Beachamp [9].

3.2.2- L'algorisme TWM

L'algorisme Two-Way Mismatch està basat en escollir la freqüència fonamental d'un instant de temps com la opció que minimitza les discrepàncies entre una suposada sèrie harmònica ideal i la real obtinguda a partir dels pics amb màxima amplitud detectats a l'espectre resultant d'aplicar la transformada de Fourier a un so (pics candidats).

El procediment consisteix en mesurar dos errors, *predicted to measured* i *measured to predicted*. El primer es defineix com:

$$Err_{p\to m} = \sum_{n=1}^{N} E_w(\Delta f_n, f_n, a_n, A_{\max}) = \sum_{n=1}^{N} \Delta f_n \cdot (f_n)^{-p} + \left(\frac{a_n}{A_{\max}}\right) \times \left[q\Delta f_n \cdot (f_n)^{-p} - r\right]$$

on Δf_n és la diferencia entre la predicció (pic ideal) i el pic més proper de la llista de pics del so, f_n i a_n són la freqüència i magnitud dels pics de la predicció i A_{max} és la màxima magnitud d'un pic.

Es calcula per cada freqüència fonamental possible la seva sèrie harmònica ideal i calculem les distàncies entre els pics d'aquesta sèrie harmònica ideal (*predicted*) i els pics més propers que tenim a la llista de candidats (*measured)*.

L'altre error *measured to predicted* es defineix com:

$$Err_{m\to p} = \sum_{k=1}^{K} E_w(\Delta f_k, f_k, a_k, A_{\max}) = \sum_{k=1}^{K} \Delta f_k \cdot (f_k)^{-p} + \left(\frac{a_k}{A_{\max}}\right) \times \left[q\Delta f_k \cdot (f_k)^{-p} - r\right]$$

on Δf_k és la diferencia entre el pic de la llista de pics del so i el pic ideal més proper, f_k i a_k són la freqüència i magnitud dels pics de la predicció i A_{max} és la màxima magnitud d'un pic.

Es calcula per cada pic candidat, la seva sèrie harmònica aproximada formada per pics de la llista de candidats a freqüència fonamental (*measured*) i calculem les distàncies entre els pics d'aquesta sèrie harmònica i la ideal formada a partir del primer pic candidat escollit (*predicted*).

L'error total, llavors és defineix com la suma ponderada dels dos anteriors:

$$Err_{total} = \frac{Err_{p \to m}}{N} + \rho \frac{Err_{m \to p}}{K}$$

i s'utilitzen uns valors pels paràmetres: p=0.5, q=1.4, r=0.5 i ρ =1 perquè experimentalment s'ha provat que funcionen amb la majoria dels sons, al punt següent veurem que en el cas del saxo, es poden ajustar per tal d'aconseguir resultats millors.

3.2.2.1- Millores sobre TWM

El procediment TWM considera que s'ha de buscar la freqüència fonamental a tot el rang possible, començant pel valor mínim i buscant per petits increments en la freqüència els possibles pics de cara a obtenir el de mínim error, però això requereix molt cost computacional i a SMS primer es busquen els possibles candidats a *pitch* i es fa la búsqueda només sobre aquests[1]. Experimentalment s'ha observat que en el cas del saxo i altres instruments de vent, les sèries harmòniques produïdes per cada nota no són exactes i depenen del registre de la nota, així les notes del registre alt solen quedar baixes d'afinació, i depèn molt de la pressió d'aire i pressió sobre la canya que exerceix

1 només es busca sobre els pics trobats a l'espectre i els seus múltiples o submúltiples sencers, o sigui la meitat de la freqüència, un terç, un quart, el doble, el triple... i aquests valors amb petits increments.

la persona que toca, que la nota quedi afinada o no, també apareix el problema de que els harmònics del so no solen tenir la mateixa amplitud en tot el registre, així les notes més greus solen tenir els primers harmònics, sobretot el fonamental, més atenuats que la resta i això passa especialment als atacs de les notes, fins que s'assoleix una afinació correcta en la part transitòria de la nota. Això porta a que hi hagi problemes en la detecció del *pitch*, que s'han resolt variant determinats paràmetres com el cas de q i ρ que en l'algorisme *TWM* prenen valors 1.4 i 1 respectivament, i s'ha decidit canviar els seus valors per 2.4 en el cas de q i 0.33 en el cas de ρ per tal de donar més importància a l'error per magnitud dels pics que no pas per desplaçament freqüencial dintre de la sèrie harmònica, precisament pel fet que s'ha dit abans de que en el saxo la sèrie d'harmònics no sol ser perfecta. També s'ha modificat el procés d'agafar els candidats a *pitch*, permetent que pics amb una amplitud no molt alta puguin formar part de la sèrie harmònica de mínim error.

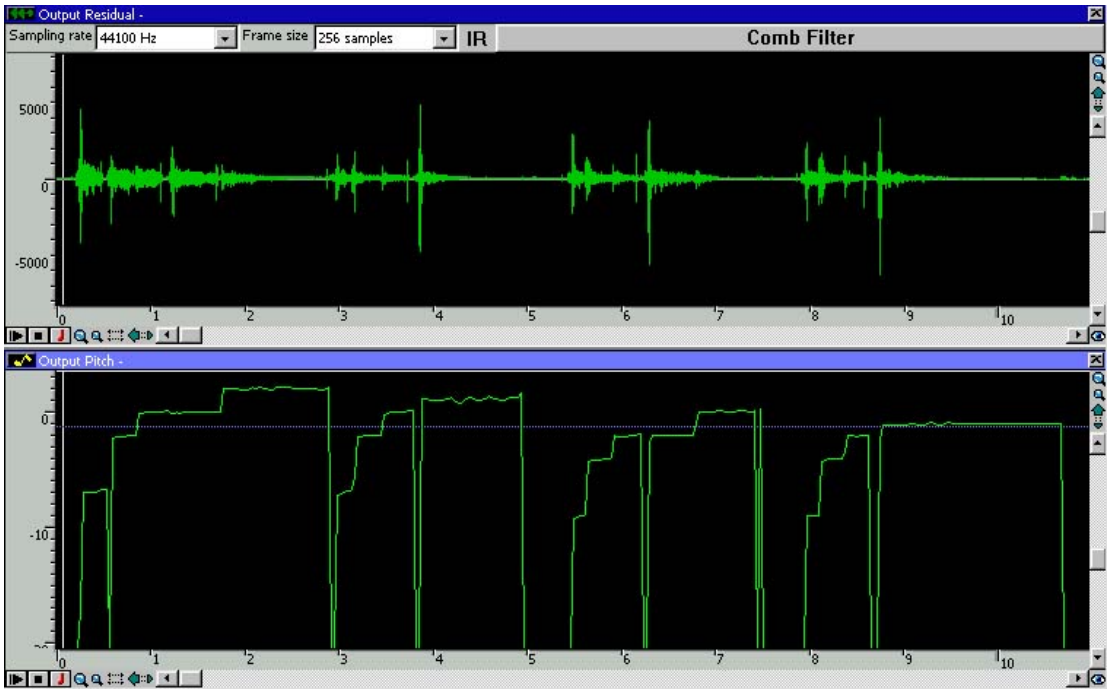


Figura 3.3: Detecció de pitch sense implementar cap millora a SMS.

En la figura 3.3, gràficament podem veure els resultats de l'anàlisi d'un fragment del tema *How High The Moon* tocat per un saxo tenor. A la part de sota podem observar l'evolució del *pitch* al llarg del temps, i a dalt la forma d'ona de la part residual del so. Si ens fixem bé podem observar que als atacs de les notes, hi ha un fort component residual, que representa el soroll que es produeix al atacar les notes, però

observem uns pics que corresponen a una dolenta detecció del pitch que fa que part del component sinusoidal del so vagi a parar a la part residual, un cop fem les millores explicades anteriorment, el següent gràfic (figura 3.4) ens mostra com els pics més importants desapareixen i la detecció és molt més acurada, permetent així una millor extracció d'atributs d'alt nivell i una millor síntesi, sense pèrdua de qualitat.

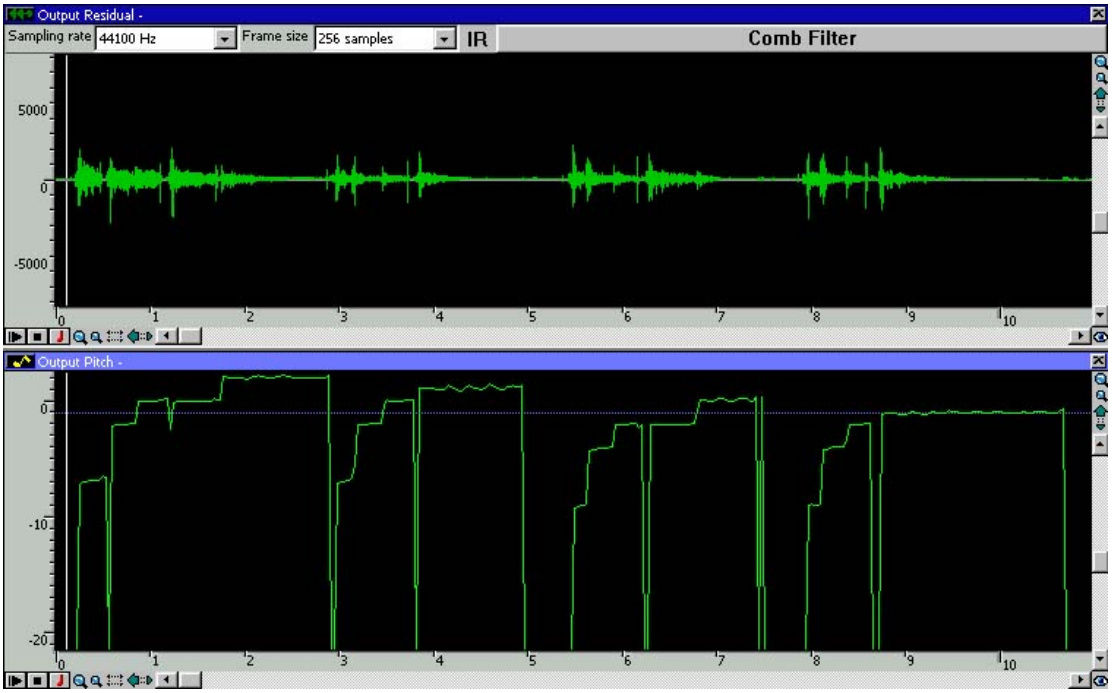


Figura 3.4: Detecció de pitch canviant el valor dels paràmetres de l'algorisme TWM.

3.2.3- Reanàlisi dels atacs (attack reanalysis)

Tot i que els resultats amb aquestes millores ja són bastant bons, encara hi ha una certa inestabilitat en l'atac de la nota que no es resol fins que passen unes mil· lèsimes de segon de l'atac i arriba la part transitòria de la nota, llavors una possible solució a aquest problema és fer servir l'opció de fer un reanàlisi de l'atac, un cop estem a una zona de la nota amb *pitch* estable, així doncs, si portem un número de *frames* analitzats considerable (en el nostre cas 15, aproximadament uns 75 mil· lisegons) amb un *pitch* constant, repetirem el procés d'anàlisi, però aquest cop anant enrera i tenint com a referència el *pitch* trobat a la part estable, així podem corregir possibles errors en canvis sobtats de nota amb intervals grans i evitar els errors que apareixien en cas de no

fer reanàlisi, que fins i tot portaven en algun cas a errors d'una octava en la decisió de la freqüència fonamental. A la figura 3.5 veiem els resultats després d'aplicar la tècnica de reanàlisi.

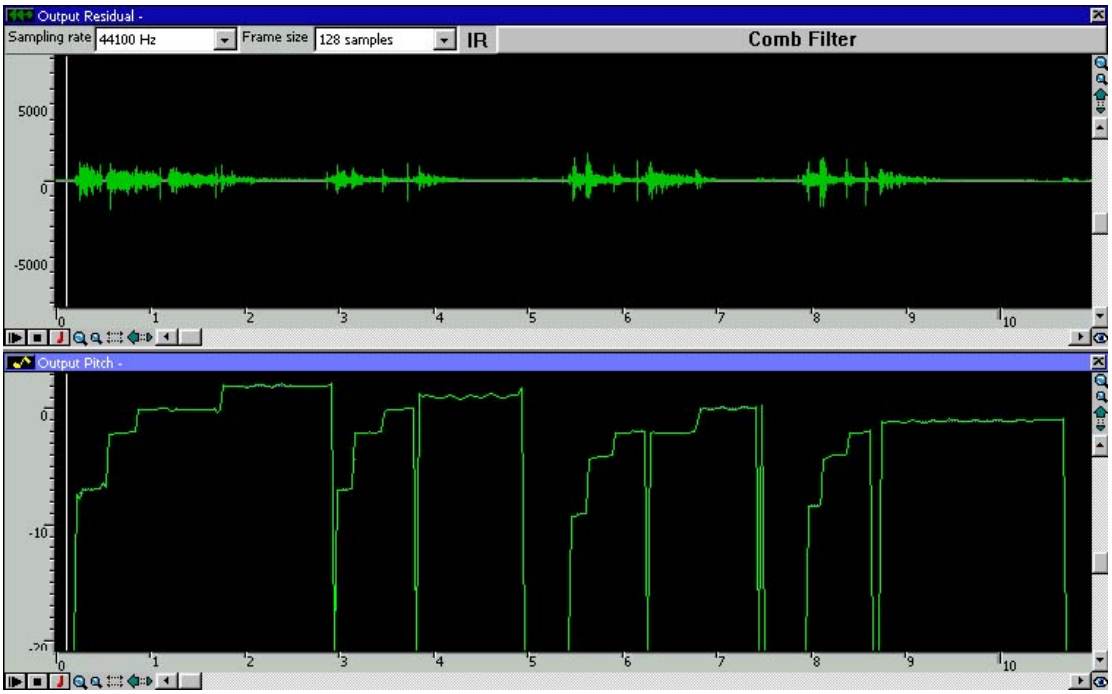


Figura 3.5: Detecció de pitch fent reanàlisi als atacs.

El problema que apareix al fer la tècnica de reanàlisi és que el rendiment en temps de computació empitjora i fa que en el cas de voler-se aplicar en aplicacions que han de funcionar en temps real tingui l'inconvenient de que introdueix una latència igual al número de *frames* que es fan servir al reanàlisi, és a dir en el cas de que es facin servir 15 frames de reanàlisi, la sortida de l'anàlisi es produirà amb una latència de com a mínim 16 frames[1]. Perquè quedi més clara la millora aconseguida en fer el reanàlisi veiem el següent exemple on hem ampliat l'atac de dues notes, a la figura de l'esquerra (figura 3.6) veiem el *pitch* detectat sense fer reanàlisi, i a la dreta (figura 3.7), fent servir un reanàlisi de 15 *frames*.

---

1 S'ha comprovat experimentalment que 15 frames són suficients per obtenir bons resultats, l'altre frame fins als 16, és la latència produïda per l'anàlisi.

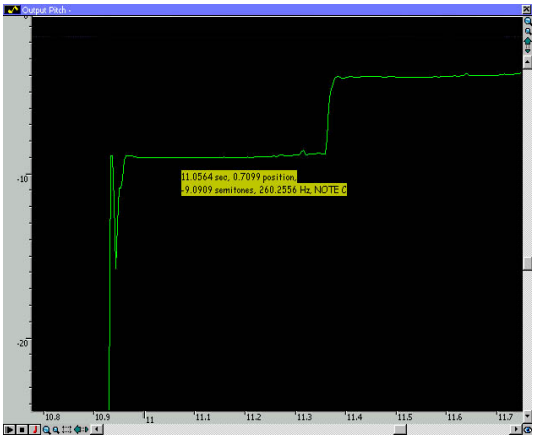


Figura 3.6:Detecció de pitch sense reanàlisi

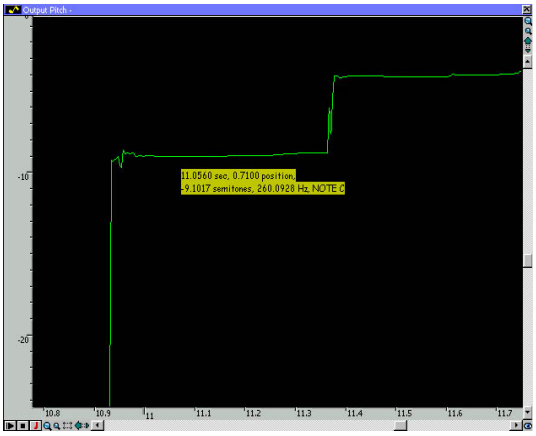


Figura 3.7: Detecció de pitch amb reanàlisi.

3.3- Segmentació en Regions

Un dels atributs d'alt nivell que més ens interessa per la finalitat del projecte és la segmentació en regions que representin les notes, és a dir, tenir una anàlisi del so on puguem identificar fàcilment les notes sense haver de treballar directament sobre instants de temps que fan que el so perdi el sentit musical que podria tenir una partitura per a un músic. Per tant el següent pas que es planteja, un cop l'anàlisi sembla que funciona correctament és segmentar el so en regions que representin les notes, i cada regió amb el seu atac, la seva part transitòria i el seu *release*.

Per fer la segmentació en notes tindrem en compte la variació en l'amplitud i freqüència (*pitch*) al llarg del temps i el coneixement musical, per descartar canvis impossibles des del punt de vista d'execució, ja que s'estan analitzant peces interpretades per un músic real, i no s'hauran d'admetre canvis que no puguin ser interpretats per un saxo. Es podria utilitzar la partitura per intentar aconseguir uns resultats potser millors, però en aquesta versió no es farà servir cap tipus de coneixement sobre la melodia que s'està analitzant, així permetem una interpretació més lliure per part del músic.

3.3.1- Variacions de l'amplitud

Definirem una nota, en funció de la seva amplitud al llarg del temps, amb les seves parts de la següent manera:

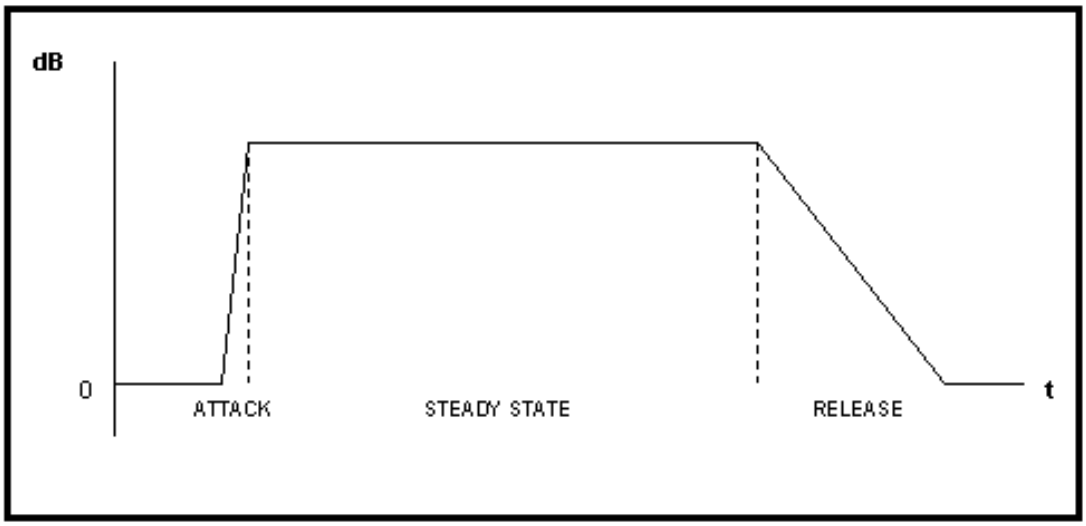


Figura 3.8: Modelització de l'amplitud de les parts d'una nota al llarg del temps.

Al gràfic (figura 3.8) veiem les parts d'una nota ideal, però a la pràctica s'ha observat que aquest model de nota no sol aparèixer molt habitualment, si bé en interpretacions no expressives el model de nota que trobem és molt semblant al de la figura, quan s'afegeix una certa expressivitat, la representació de la nota varia de forma considerable, per exemple un recurs expressiu molt utilitzat és afegir *vibrato*[1] a la nota de tal forma que la part transitòria de la nota no té una amplitud constant, sinó que fa petites oscil· lacions al llarg del temps com es pot veure al següent gràfic (figura 3.9).

---

1 Com veurem més endavant, el vibrato són petites oscil· lacions a l'amplitud o freqüència de la nota, a vegades també es parla de tremolo o bending referint-se a variacions en la freqüència, però això és més comú en instruments com la guitarra on hi ha una palanca destinada a aquesta funció. En un saxo l'efecte més comú és el vibrato d'amplitud, que correspon a oscil· lacions en el volum de la nota produïdes pels canvis de pressió de l'aire que expulsa el músic.

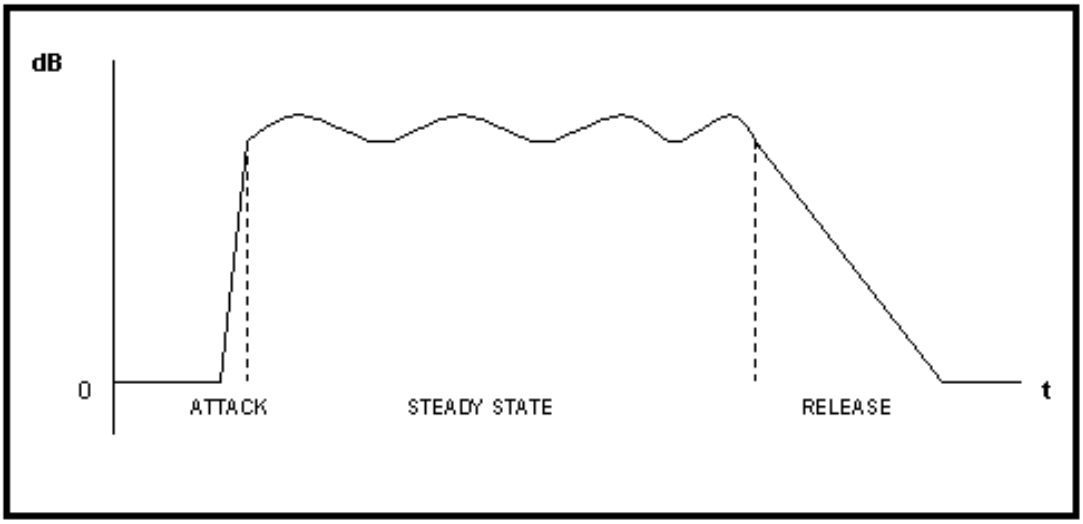


Figura 3.9: Modelització de les parts d'una nota produïda amb vibrato.

Altres situacions amb les que ens trobem són casos en que una nota no té atac, bé perquè la nota es toca lligada amb l'anterior, suprimint en aquest cas el descens en l'amplitud a la nota anterior, fenòmen que es coneix amb el nom de *legatto* (veure figura 3.10) o bé perquè s'ataca amb molt poca energia, aconseguint un so molt més dolç. En el cas de trobar un *legatto* pot ser que les dues notes tinguin freqüències diferents o la mateixa.

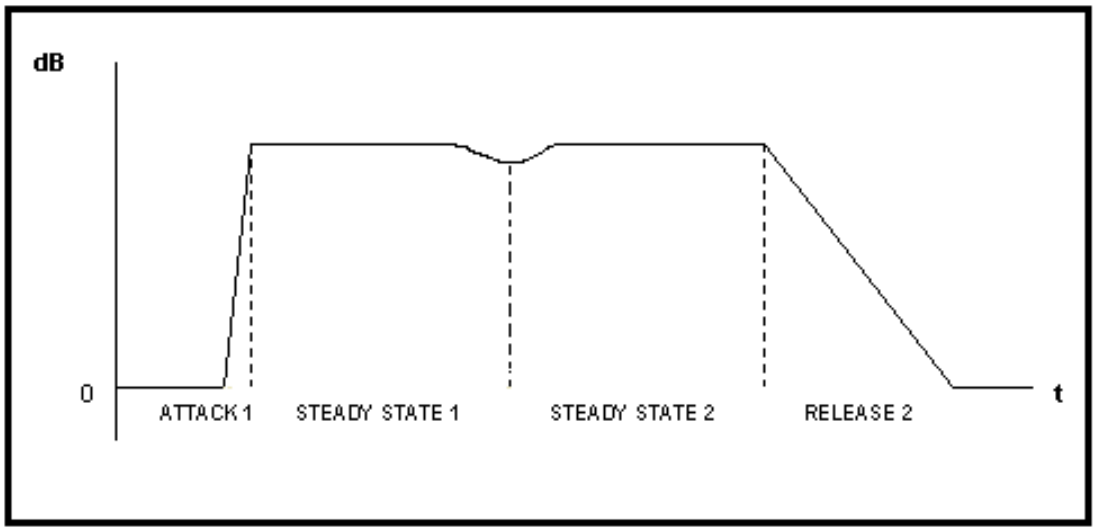


Figura 3.10: Dues notes lligades sense atac a la segona i sense *release* a la primera (legatto).

També trobem casos en que la part final de la nota manté la mateixa energia i volum que la part transitòria, desapareixent en aquest cas la regió *release* com es veu a la figura 3.11.

Generalment la part transitòria o *steady state* sol durar molt més que les altres regions de la nota, però hi ha una forma d'articulació que s'ha de tenir present, anomenada *stacatto* en qual la part transitòria dura poc més que l'atac i on la regió *release* no apareix (figura 3.12).

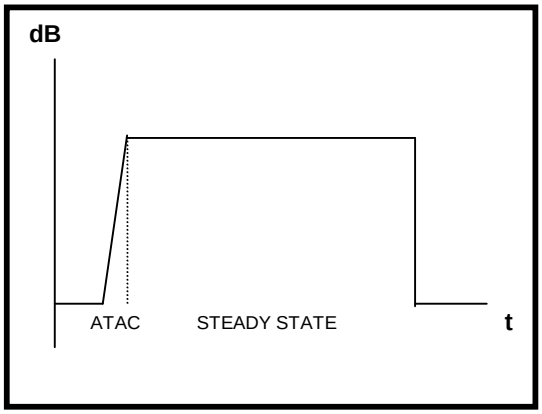


Figura 3.11: Nota sense *release*

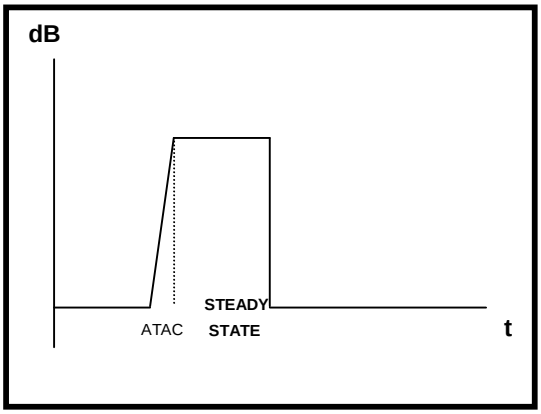


Figura 3.12: Nota produïda amb *Stacatto*.

3.3.2- Variacions de la freqüència

Si analitzem el *pitch* de la nota, observarem que la nota assoleix la seva freqüència que la caracteritza a l'atac, amb petites variacions degut al soroll produït pel canvi de nota o l'articulació, i manté la freqüència durant tota la nota, sempre i quan estigui afinada durant tota l'execució i no s'introdueixin tècniques expressives, com per exemple començar la nota a una freqüència menor i pujar-la durant l'atac o fins i tot durant part de la regió estable de la nota fins a assolir l'afinació desitjada, efecte que es coneix amb el nom de *glissando* (figura 3.15), o introduir *vibrato* al igual que el d'amplitut explicat abans però amb petites oscil· lacions a la freqüència (figura 3.14).

Veiem aquests casos, primerament la situació ideal (figura 3.13):

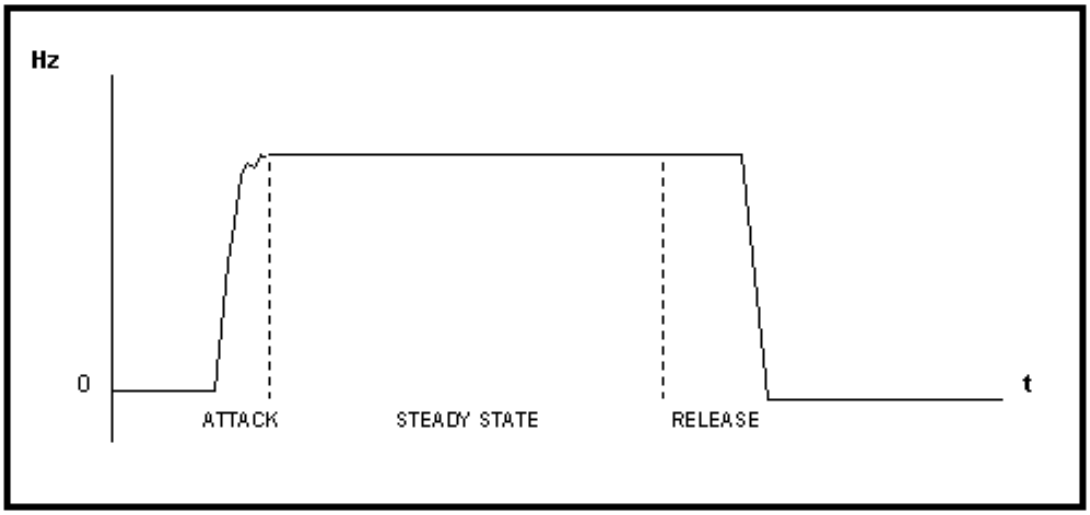


Figura 3.13: Modelització ideal de la freqüencia d'una nota al llarg de les seves parts.

I algunes possibles variacions expressives:

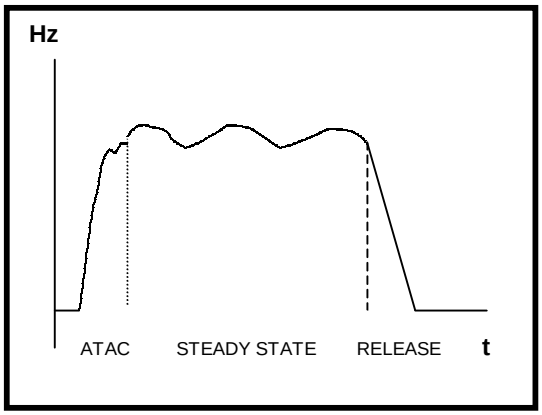


Figura 3.14: Nota amb vibrato de freqüència

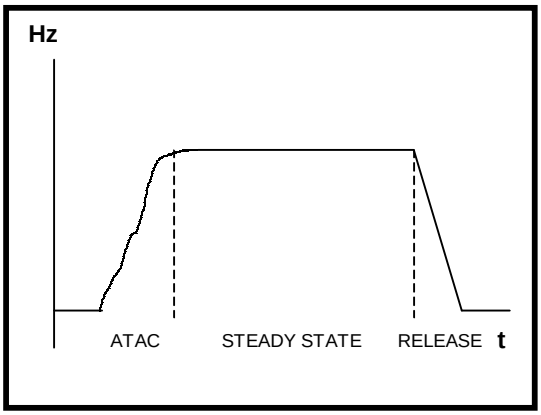


Figura 3.15: Nota amb glissando a l'atac.

3.3.3- Tipus d'articulacions

Com hem vist les possibilitats amb les que ens podem trobar són moltes i això repercutirà molt en la dificultat per fer la segmentació en regions de les melodies, sobretot pels casos expressius. Llavors, cal analitzar les possibles articulacions que es poden produir en els canvis de nota. En el cas d'un saxo i en general en tots els instruments de vent, la transició entre dues notes es pot produir:

- anant d'una nota a una altre, sense atac a l'inici de la segona i generalment sense *release* a la primera nota, que hem definit com *legatto*.

- lligades una darrera de l'altre amb atac a la segona de les notes.

- sense lligar, és a dir notes separades, amb silenci entre mig.

Per la particularitat de ser un instrument monofònic, mai ens podem trobar amb la possibilitat de que una nota se sobreposi a l'altre, és a dir, que comenci una nota abans de que acabi l'altre, com podria passar a un violí.

Al fer la segmentació en regions d'interpretacions no expressives, els casos amb que ens trobem són menys complexes que pels casos amb expressivitat, ja que totes les notes són molt similars al cas ideal, tenint el seu atac, regió estable i final de la nota molt definits, i no apareixen situacions especials com l'stacatto, el vibrato o el glissando.

Veiem la segmentació en regions feta sobre una interpretació no expressiva del començament del tema Autumn Leaves interpretat per un saxo tenor (autum-q0.snd) (figura 3.16). A la finestra de dalt es veu la variació de la freqüència fonamental al llarg del temps i a la finestra de baix, la variació de l'amplitud al llarg del temps, les regions estan separades entre si per línies discontínues vermelles i el tipus de regió s'indica a dalt amb un color, així els atacs estan representats pel color taronja, les regions estables pel color groc i els finals de nota pel color blanc:

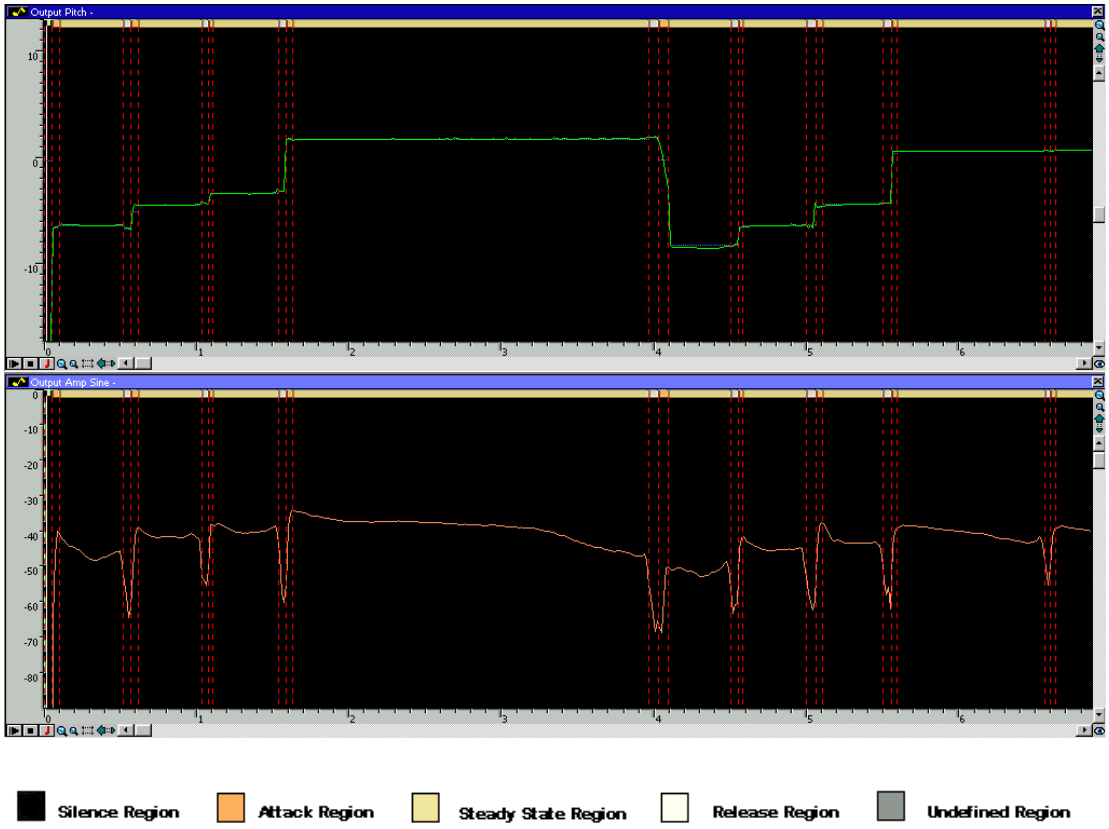


Figura 3.16: Segmentació en regions d'un fragment del tema Autumn Leaves interpretat de forma no expressiva.

Si observem a la figura 3.17 la mateixa frase del gràfic anterior interpretada expressivament (autum-e1.snd), veiem que les variacions en la freqüència i l'amplitud són molt més exagerades, i la detecció de les regions es fa molt més complexa, ja que en alguns casos els efectes de *vibrato* o *glissando* fan molt difícil saber si realment s'està canviant de nota o si es tracta d'un recurs que ha fet servir el músic que ha interpretat la melodia per afegir expressivitat al tema. Observem que a la primera nota trobem un clar exemple de *glissando* ja que fins a gairebé la meitat de la nota no s'assoleix la freqüència que la caracteritza. També entre les notes 5 i 6 veiem un exemple de *legatto*, barrejat amb un *vibrato* d'amplitud bastant exagerat, que s'accentua sobretot a la darrera nota, on fins i tot la freqüència es veu afectada decreixent lleugerament a la part mitja de la regió estable de la nota. Fixem-nos que malgrat els múltiples recursos expressius utilitzats a l'interpretació, la detecció de les regions és molt acurada.

Al següent punt veurem com s'ha implementat l'algorisme encarregat de fer la segmentació en regions i la decisió del tipus de cada regió.

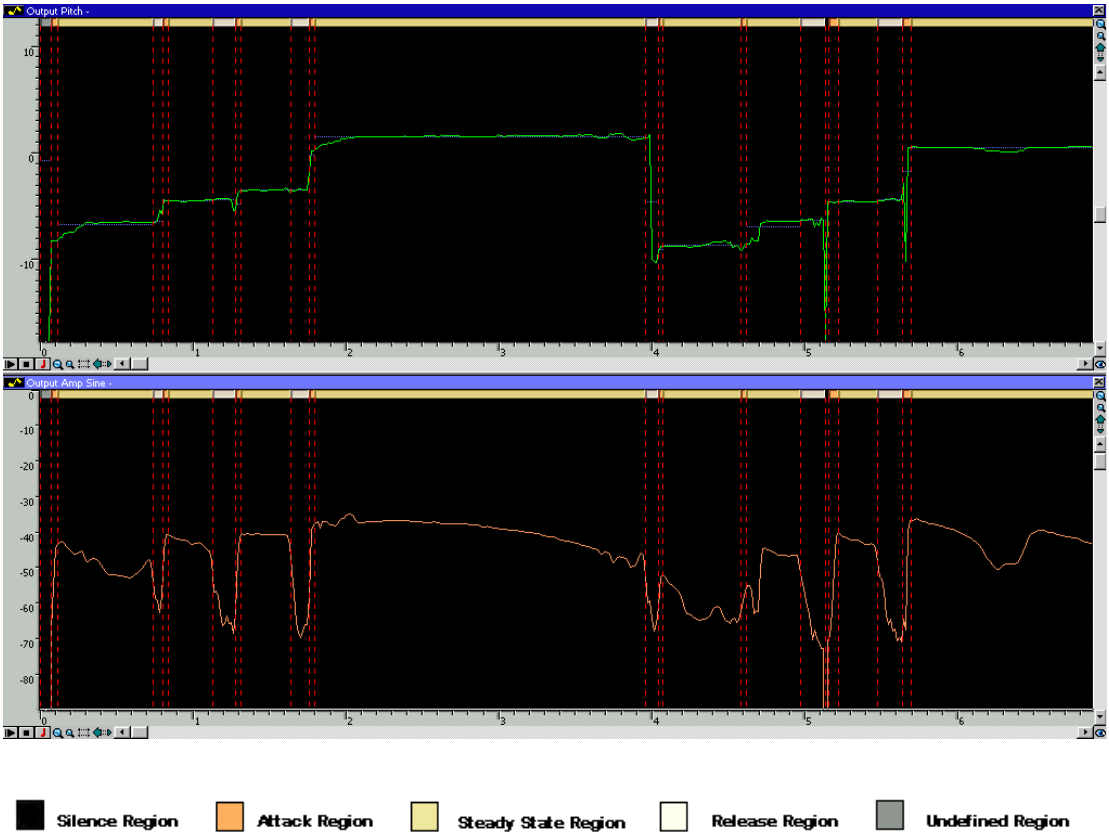


Figura 3.17: Segmentació en regions d'un fragment del tema Autumn Leaves interpretat de forma expressiva.

3.3.4- Algorisme de detecció de regions

En aquest capítol, explicarem en detall els passos seguits a l'algorisme per tal de fer la segmentació del so original en les regions musicals definides com a *attack*, *steady state*, *release* i *silenci*.

L'algorisme fa la detecció de les regions en temps real[1] en el sentit de que no fa cap tipus de *backtracking* a l'hora de decidir on comença o acaba cada regió o de quin

---

1 quan parlem de temps real no ens referim a temps de computació, ja que en aquest cas dependria de la potencia de la màquina que es fa servir, sinó de la latència amb que s'obtenen els resultats, d'aquesta manera la particularitat de l'algorisme és que produeix els resultats a mesura que s'està analitzant el fitxer de so, sense produir cap retard, per tant la latència és mínima.

tipus és cada regió detectada, per tant es fa la decisió en base només a canvis produïts en instants de temps anteriors al punt que s'està analitzant. Tot i que per la finalitat de SaxEx no era necessari que l'algorisme funcionés en temps real i que potser es podria haver dissenyat una detecció de regions independent del procés d'anàlisi SMS que treballés sobre els resultats produïts un cop ja s'ha analitzat un so, es va creure molt important poder dissenyar un algorisme que funcionés en temps real i que estigués integrat dintre del procés SMS per així poder ser utilitzat en futures aplicacions SMS d'anàlisi, transformació i síntesi de so en temps real per aconseguir, per exemple, reproduir amb una certa expressivitat una melodia que s'està interpretant sense gens d'expressivitat.

L'algorisme fa servir només informació de l'anàlisi SMS referent a la freqüència i l'amplitud del so en la seva part sinusoidal, i encara que seria molt útil fer servir informació sobre la partitura del tema que s'està analitzant, no s'ha cregut convenient utilitzar aquesta informació, per així permetre a l'intèrpret de la melodia que s'analitza, una llibertat més amplia en quan al *tempo* o *rubatto*[1] i l'expressivitat que li vulgui donar, i no estar limitats a interpretacions on l'artista segueix la partitura estrictament al peu de la lletra.

Hem de tenir present que sense informació conjunta dels canvis en la freqüència i amplitud del so, seria pràcticament impossible fer la segmentació, ja que ens podem trobar amb casos en que canviem de nota però no canviem de freqüència i casos en que canviem de nota però l'amplitud es manté gairebé constant. Una primera aproximació va ser fer servir la freqüència representada en *Hertz*, però més endavant es va veure que era més aconsellable utilitzar una representació mitjançant altres unitats anomenades cents[2] que tenen més sentit quan s'analitzen melodies per detectar canvis de nota. El problema apareixia al decidir, per exemple, que un canvi en la freqüència de 15 Hz eren suficients per dir que s'havia produït un canvi de nota per un semitò, això era veritat al

---

1 El tempo està relacionat amb la velocitat a la qual és interpretada la cançò, es defineix en unitats de notes negres per minut i el rubatto és un recurs expressiu molt utilitzat en el jazz que consisteix en no limitar-se a la durada exacta de les notes, sino, allargar o comprimir lleugerament les notes a gust de l'interpret per donar una certa inestabilitat a la melodía.

2 els cents són una unitat per representar la freqüència basada en dividir una octava musical en 1200 parts/cents (12 semitons), d'aquesta manera dues notes musicals separades per un interval igual, suposem una tercera major , sempre tindran la mateixa separació, en aquest cas 408 cents, independentment de l'octava a la que estan, en canvi en una representació mitjançant Hertz, la quantitat d'Hertz entre les dues notes depèn de l'octava on estan, havent-hi més separació a les octaves superiors. En la representació mitjançant cents, el zero equival a la nota A 440Hz.

voltant dels 440Hz, però quan es produïa un canvi de 15Hz al voltant dels 150Hz, el canvi era d'un to, i a les freqüències altes, al voltant dels 800 Hz, un canvi de 15 Hz es pot interpretar com un vibrato de freqüència, ja que per haver un canvi d'un semitò, ha d'haver com a mínim un canvi en la freqüència de 50 o 60 Hz. Amb els cents, aquest problema desapareix completament, ja que un canvi d'un semitò seran sempre 100 cents, independentment de la freqüència de les notes que intervenen en el canvi.

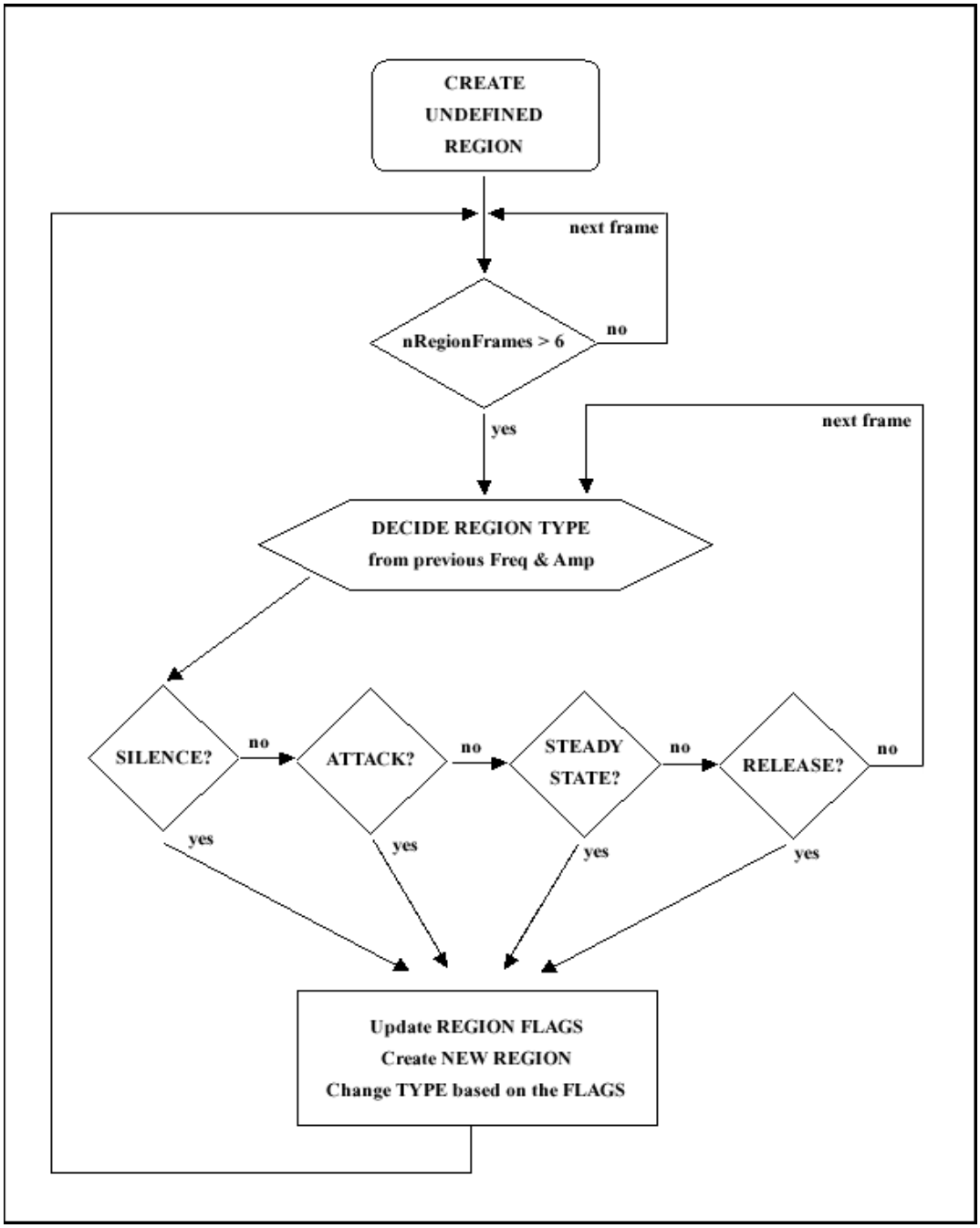


Figura 3.18: Diagrama de blocs de l'algorisme de detecció de regions.

El procediment utilitzat és el següent (veiem figura 3.18), primer es crea una regió de tipus indeterminat, ja que a l'inici de l'anàlisi no tenim informació anterior i no podem decidir el tipus de regió a la que estem (generalment a l'inici de l'anàlisi trobarem silenci, així que la següent regió que es crearà serà generalment silenci).

Llavors, un cop portem analitzats 6[1] frames, anirem mirant a frames anteriors l'evolució de la freqüència i l'amplitud i depenent de la situació en que ens trobem, activarem els flags de tipus de regió i, si trobem que s'han produït variacions tal que es poden modelar com un canvi musicalment possible, activem també el flag de crear una nova regió. Els canvis musicalment possibles són els que es poden descriure com alguna de les articulacions o transicions entre notes explicades abans. Veiem casos impossibles musicalment parlant de canvis de regió:

- *una regió release després d'un silenci o d'un atac*: ha de tenir almenys una part estable abans. Després d'un silenci s'espera un atac, en els casos no expresius, o una regió estable directament, en el cas d'interpretacions expressives tocades amb una certa docilitat o suavitat.

- *una regió silenci després d'un atac*: en aquest cas es considera la regió atac també com a silenci, perquè serà possiblement un soroll de gravació. Després d'un atac, sempre s'espera una regió estable.

- *una regió atac després d'una regió estable de menys d'un determinat nombre de frames*: suposarem que perquè una regió sigui considerada com a estable, haurà de tenir un cert nombre de frames (25 és una quantitat acceptable ~ 50ms), si ens trobem amb aquest cas el considerarem com un *vibrato* exagerat.

Llavors per a cada frame que es va analitzant, si s'ha activat el flag crear una nova regió, s'assigna a la regió actual el tipus que s'ha detectat abans, s'omple la informació sobre la regió com és la durada i el temps de començament, i es crea una nova regió, seguint el mateix procediment que s'ha explicat. Veiem a la figura 3.18 un esquema per entendre millor l'algorisme utilitzat i a la figura 3.19 el pseudocodi de l'algorisme.

---

1 aquest número és pot ajustar segons el tipus d'interpretació que s'està analitzant, però experimentalment s'ha comprovat que el valor 6 funciona bastant bé tant per interpretacions inexpressives com expressives, fa referència al número de frames anteriors que s'utilitzen per decidir a quina regió estem en cada moment. Per interpretacions inexpressives, un valor menor (4 o 3) també donaria bons resultats

```
if (pNewRegion->nFrames > 6) // si la regió actual te més de 6 frames...
{	//SILENCI
	if	(LastRegionIsRelease || LastRegionIsSilence || LastRegionIsAttack || LastRegionIsSS >= 1)
		&&
	(Pitch de frames anteriors == 0 && Amplitud == 0)
	{
		if	( es compleixen les condicions de regio silenci per l'actual frame)
		{
			if (LastRegionIsAttack) //serà un soroll, hauria de ser silenci
			{
				pNewRegion->RegionType = Region_type_SILENCE;
				LastRegionIsSilence = 1;
				LastRegionIsAttack = 0;
				LastRegionIsSS = 0;
				LastRegionIsRelease = 0;
			}

			if (LastRegionIsSilence==0) //si la regió anterior no era silenci, aquesta si ho serà
			{
				NewRegion = TRUE;
				FirstRegion = 0;
				LastRegionIsSilence = 1;
				LastRegionIsAttack = 0;
				LastRegionIsSS = 0;
				LastRegionIsRelease = 0;
			}

			LastRegionIsSilence = 1;
		}
	}
	else if ( tenim pitch > 0)
	{
		//ATTACKS
		if (LastRegionIsRelease || LastRegionIsSilence || LastRegionIsSS >= 25) //un steady state ha de durar
més de 30 frames
		  &&
		  (en els 6 ultims frames l'amplitud ha pujat amb pendent gran)
		{
			if (LastRegionIsAttack ==0)
			{
				NewRegion = TRUE;
				FirstRegion = 0;
				LastRegionIsSilence = 0;
				LastRegionIsAttack = 1;
				LastRegionIsSS = 0;
				LastRegionIsRelease = 0;
			}
			LastRegionIsAttack = 1;
		}
		else
```

```
//STEADY STATES
if (LastRegionIsAttack || LastRegionIsSS >=2) //la regio anterior era un attack o un SS (legato)
  &&
  (la variació de l'amplitud en valor absolut es manté constant)
  &&
  (la variació del pitch en valor absolut es manté constant)
{
        if (LastRegionIsSS ==0)
        {
                NewRegion = TRUE;
                FirstRegion = 0;
                LastRegionIsSilence = 0;
                LastRegionIsAttack = 0;
                LastRegionIsSS = 1;
                LastRegionIsRelease = 0;
        }
        LastRegionIsSS++; //incrementa el número de frames de la regio steady state
}
else
//RELEASES
if (LastRegionIsSS >= 30) //si la regió anterior era un steady state amb mes de 30 frames
  &&
  (en els 6 ultims frames l'amplitud ha baixat considerablement)
  &&
  (la variació del pitch en valor absolut es manté constant)
{
        if (LastRegionIsRelease ==0)
        {
                NewRegion = TRUE;
                FirstRegion = 0;
                LastRegionIsSilence = 0;
                LastRegionIsAttack = 0;
                LastRegionIsSS = 0;
                LastRegionIsRelease = 1;
        }
        LastRegionIsRelease = 1;
}
else
//LEGATOS
if (LastRegionIsSS >= 30) //si la regió anterior era un steady state amb mes de 30 frames
  &&
  (en els útims 6 frames, el pitch ha variat considerablement)
{
        NewRegion = TRUE;
        FirstRegion = 0;
        LastRegionIsSilence = 0;
        LastRegionIsAttack = 0;
        LastRegionIsSS = 1;
        LastRegionIsRelease = 0;
        LastRegionIsSS++; //incrementa el número de frames de la regio steady state
}
    }
}
```

```
//CREACIÓ DE NOVES REGIONS
if ( NewRegion || FirstRegion ) //s'ha detectat una nova regió o estem a la primera...
{
        (actualitzem la durada i temps d'inici de la regió);

        if (LastRegionIsSilence)
        {
                if ( la regió anterior es un steady state que dura menys de 0.1 segons)
                {
                        //la regio anterior sera també silenci, serà un soroll
                        (canviem el tipus de la regió anterior per silenci);
                        (creem la nova regió de tipus silenci);
                }
                else
                {
                        (creem la nova regió de tipus silenci);
                }

        } else if (LastRegionIsAttack)
        {
                (creem la nova regió de tipus atac);

        } else if (LastRegionIsSS >= 2)
        {
                (creem la nova regió de tipus steady state);

        } else if (LastRegionIsRelease == 1)
        {
                if (pNewRegion->Duration > 0.1)
                {
                        (creem la nova regió de tipus release);
                }
                else //si la regió d'abans del release, steady state dura menys de 0.1 segons es un soroll fortuit
                {
                        //Steady State és realment Silenci
                        (canviem el tipus de la regió anterior per silenci);
                        (creem la nova regió de tipus silenci);
                }
        }
}
```

Figura 3.19: Pseudocodi de l'algorisme de segmentació en regions musicals.

S'ha provat la detecció en regions per cinc melodies de temes estàndards de jazz (All of Me, Autumn Leaves, How High the Moon, Misty i My One and Only Love), i s'han fet servir per a cada tema 4 o 5 interpretacions diferents, sense expressivitat i amb diferents graus i qualitats d'expressivitat. El més important era obtenir uns resultats

molt acurats pels casos inexpressius, ja que seran el *input* del sistema SaxEx, i els resultats obtinguts dependran molt dels resultats de l'anàlisi i de la segmentació en regions que es faci, els casos expressius també són importants perquè són els que s'analitzaran per tal d'incloure exemples expressius a la memòria de raonament de SaxEx, però aquest és un procés d'entrenament del sistema que pot ser supervisat per l'administrador del raonador, que pot corregir els possibles errors en la detecció de regions.

Els resultats obtinguts per l'anàlisi dels casos no expressius, és molt bo, arribant al 99 % d'encert[1] en la majoria dels casos analitzats. Al capítol de proves (capítol 5), es veuran unes taules on es mostren els resultats en detall i es parla dels casos en que l'algorisme falla.

Pels casos expressius, els resultats també són molt bons, tot i la complexitat d'algunes articulacions que fa servir l'intèrpret. Al capítol de proves (capítol 5), també es veuran unes taules on es mostren els resultats en detall dels casos expressius, i es parla de possibles pre-processos per tal de millorar la segmentació en regions.

3.3.5- Representació de les regions

Al procés d'anàlisi SMS, un cop es detecten les regions, es van gravant a unes estructures de memòria amb informació sobre el *pitch*, tipus, començament, durada, amplitud,... que després seran gravades a disc en cas de que l'usuari ho demani. Veiem com s'organitzen aquestes estructures i com s'interrelacionen entre sí.

Els fragments d'àudio que s'analitzen estan representats internament a SMS per una superclasse anomenada CChunk, aquesta classe a la seva vegada està formada per una sèrie de frames individuals representats per la classe CSMSFrame, i tots aquests frames s'agrupen a dintre d'una classe més gran anomenada CSMSFrameCollection. La

---

1 el percentatge d'encert fa referència a les notes que es detecten correctament en el sentit que els resultats aportaran una entrada bona per SaxEx, de cara a després poder treballar fent transformacions del fitxer de só basades en contingut, és a dir poder treballar sobre les notes musicals directament. Es considerarà una mala detecció quan dues notes siguin considerades com una de sola o es detecti un atac on no hi ha un atac, o una nota es consideri silenci,...

classe on es gravaran les regions, anomenada CSMSRegion, serà llavors una classe semblant a CSMSFrameCollection, ja que també contindrà una sèrie de frames, però tindrà atributs i funcions pròpies d'aquesta classe, que no tindrien sentit a la classe mes genèrica CSMSFrameCollection, per tant CSMSRegion serà una classe derivada de CSMSFrameCollection. Els atributs que ens interessen de la classe CSMSRegion per la finalitat del projecte serán el tipus de la regió (RegionType), la mitjana del pitch de la regió (Pitch), el nom de la nota (mNoteId) i altres atributs de la classe més genèrica com el número de frames (nFrames), la durada (Duration) o el apuntador a la llista d'objectes de la classe CSMSFrame (**pFrameArray). Veiem un esquema (a la figura 3.20) on queda més clara la estructura de la classe que guarda les regions i la interrrelació entre les classes que guarden els frames a SMS, a la figura només s'especifiquen els atributs que poden tenir sentit en el desenvolupament d'aquest projecte ja que, per exemple, la classe CSMSFrame té molts més atributs i funcions membre per finalitats molt diverses com el reconeixement de la parla o anàlisi d'altres tipus d'instrument, entre aquests atributs trobem informació sobre els parcials i residu de l'anàlisi, dinàmiques, fase o energia del frame.

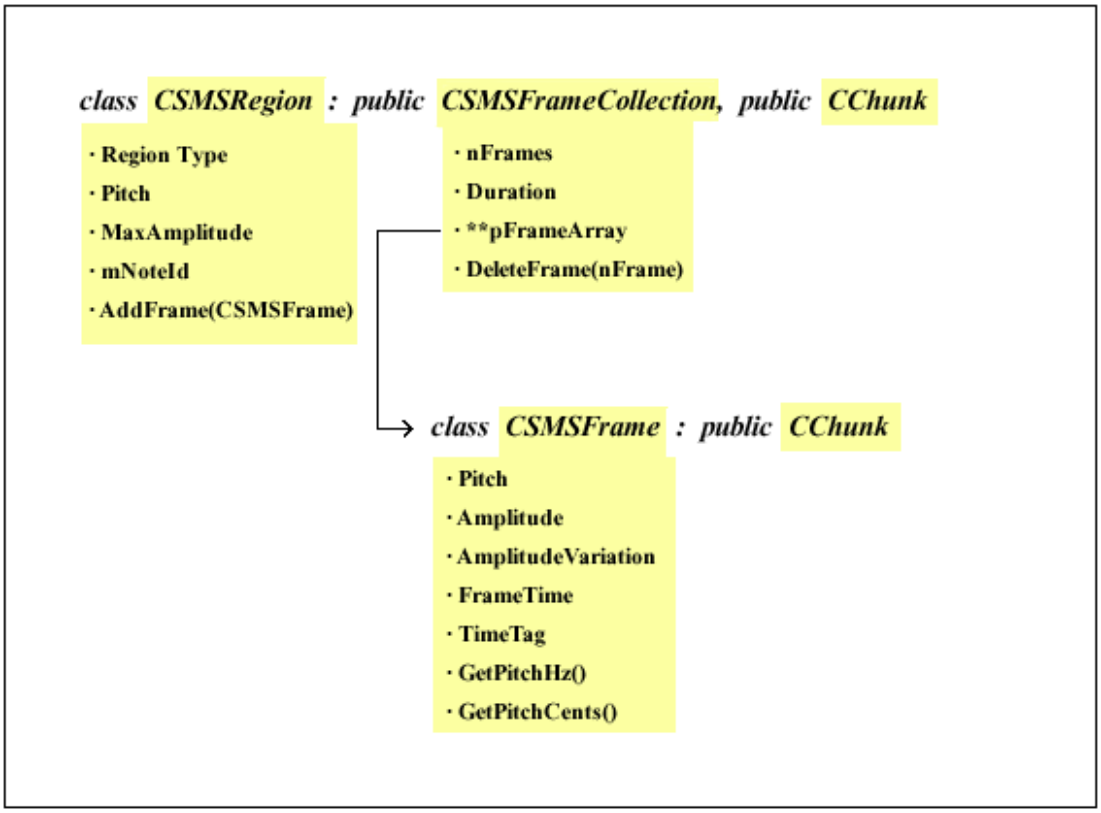


Figura 3.20: Esquema de la classe CSMSRegion, encarregada de guardar les regions en SMS.

Internament a SMS hi ha un apuntador anomenat pSMSGenericTrack que conté un array amb totes les regions del fitxer, així podem accedir a qualsevol regió i consultar les seves dades de la següent manera:

```
pSMSGenericTrack->pSMSRegionArray[i].memberVar;
```

on *i* és el número de la regió i *memberVar* és la variable que volem consultar de la regió *i*.

També és possible consultar, seguint l'esquema de la figura 3.20, per exemple el pitch en Hertzs del frame *f* de la regió *k* de la següent manera:

```
pSMSGenericTrack->pSMSRegionArray[k]->pFrameArray[f]->GetPitchHz();
```

amb aquesta estructura de classes podem accedir en cada moment a qualsevol dada de les regions o dels frames individuals d'una forma clara i entenedora.

A part de la representació de les regions a dintre de l'estructura d'anàlisi SMS, s'ha inclòs un paràmetre d'anàlisi anomenat *SaveRegionsFile* de tipus string que conté el path d'un fitxer on es gravarà només la informació sobre les regions que serà útil pel procés de raonament que farà SaxEx. Aquest fitxer tindrà extensió *.srg i inclourà la següent informació (veure figura 3.21):

- *Número total de regions del fitxer*: la primera línia del fitxer conté el número total de regions existents al fitxer.

- *Informació sobre les regions*: Tenim per cada regió una línia amb la següent informació: Numero;Tipus;Començament;Durada;Pitch;NumNota

    - *Numero*: és el numero de regió començant desde zero i en sentit ascendent.

    - *Tipus*: és el tipus de la regió, pot tenir 4 valors possibles, 0 voldrà dir silenci, 1 atac, 2 regió estable, 3 *release* i 4 indefinida (generalment als primers frames d'un fitxer, quan encara no tenim informació pasada).

- *Començament*: és l'instant en segons respecte al començament del fitxer, on comença la regió.

- *Durada*: durada de la regió, també en segons.

- *Pitch*: només per les regions estables, es guarda el càlcul de la freqüència mitja de la regió, en Hertz. Per altre tipus de regions no té sentit, ja que no es manté una freqüència fixa durant la regió.

- *NumNota*: és el número de nota a la que pertany la regió. D'aquesta manera, les regions *attack*, *steady state* i *release* d'una mateixa nota tindran el mateix número. En el cas de que una regió sigui del tipus silenci o indefinit, se li assigna el número de nota zero, ja que aquestes regions no pertanyen a cap nota. Així els números de nota comencen a 1 i s'incrementen en sentit ascendent.

```
59 Regions
0;4;0.000000;0.020317;0.000000;0
1;0;0.020317;0.023220;0.000000;0
2;1;0.043537;0.037732;0.000000;1
3;2;0.081270;0.505034;484.965210;1
4;3;0.586304;0.058050;0.000000;1
5;1;0.644354;0.043537;0.000000;2
6;2;0.687891;0.304762;363.142456;2
7;3;0.992653;0.060952;0.000000;2
8;1;1.053605;0.031927;0.000000;3
9;2;1.085533;1.915646;302.292145;3
10;0;3.001179;0.020317;0.000000;0
11;0;3.021497;0.467302;0.000000;0
12;1;3.488798;0.040635;0.000000;4
13;2;3.529433;0.336689;483.484039;4
14;3;3.866122;0.055147;0.000000;4
15;1;3.921270;0.029025;0.000000;5
16;2;3.950295;0.290249;541.014526;5
17;3;4.240544;0.046440;0.000000;5
18;1;4.286984;0.037732;0.000000;6
19;2;4.324717;0.272834;483.953888;6
20;3;4.597551;0.058050;0.000000;6
21;1;4.655601;0.031927;0.000000;7
22;2;4.687528;0.446984;456.366821;7
23;3;5.134512;0.046440;0.000000;7
24;1;5.180952;0.020317;0.000000;8
25;2;5.201270;0.339592;385.188660;8
26;3;5.540862;0.060952;0.000000;8
27;1;5.601814;0.026122;0.000000;9
28;2;5.627937;3.091156;304.493439;9
29;0;8.719093;0.020317;0.000000;0
30;0;8.739410;0.542766;0.000000;0
31;1;9.282177;0.043537;0.000000;10
32;2;9.325714;0.786576;408.775055;10
33;3;10.112290;0.052245;0.000000;10
34;1;10.164535;0.031927;0.000000;11
35;2;10.196463;0.200272;365.007751;11
36;3;10.396735;0.066757;0.000000;11
37;1;10.463492;0.029025;0.000000;12
38;2;10.492517;1.628299;302.715332;12
39;3;12.120816;0.026122;0.000000;12
40;1;12.146939;0.066757;0.000000;13
41;2;12.213696;0.458594;287.288330;13
42;3;12.672290;0.046440;0.000000;13
43;0;12.718730;0.020317;0.000000;0
44;1;12.739048;0.026122;0.000000;14
45;2;12.765170;0.304762;302.573853;14
46;3;13.069932;0.040635;0.000000;14
47;1;13.110567;0.040635;0.000000;15
48;2;13.151202;0.325079;395.211426;15
49;3;13.476281;0.034830;0.000000;15
50;1;13.511111;0.020317;0.000000;16
51;2;13.531429;0.325079;405.463715;16
52;3;13.856508;0.046440;0.000000;16
53;1;13.902948;0.031927;0.000000;17
54;2;13.934875;1.082630;363.457367;17
55;3;15.017506;0.043537;0.000000;17
56;1;15.061043;0.063855;0.000000;18
57;2;15.124898;3.018594;325.117035;18
58;3;18.143492;0.034830;0.000000;18
```

Figura 3.21: Exemple de fitxer SaveRegions (allme_q1A.srg).

Capítol 4: ScoreMan

4.1- Què és ScoreMan?

ScoreMan és un software dissenyat expressament per aquest projecte per a fer de connexió entre SaxEx i SMS, el que es pretén és, a partir de la sortida generada per SaxEx, que en aquest cas és un fitxer ascii amb els canvis que s'han de fer sobre el so inexpressiu original, i amb el fitxer amb la informació de les regions del so, resultant de l'anàlisi SMS, fer una conversió i generar un nou fitxer (score de síntesi) que entengui SMS i fer la crida per tal de que SMS sintetitzi el nou so expressiu. S'ha dotat a ScoreMan d'una interfície gràfica implementada en Visual C++ [10] [11] utilitzant les Microsoft Foundation Classes (MFC), per la seva facilitat a l'hora de fer menús i per fer servir la filosofia document-vista a l'hora de representar interna i visualment els diferents fitxers que es manegen al programa. El fet d'utilitzar una interfície gràfica permet a l'usuari del programa, carregar i visualitzar els fitxers de sortida de SaxEx i amb un complet sistema de menús, generar el fitxer per fer la síntesi, així com visualitzar d'una manera llegible els fitxers que contenen les regions del fitxer que s'ha

analitzat i salvar a disc o imprimir qualsevol d'aquests fitxers. A més, aquest programa accepta la possibilitat de configurar-lo com a servidor acceptant, mitjançant un socket obert vinculat a un port determinat, peticions remotes per sintetitzar el nou so, o qualsevol altre ordre possible del programa que s'especifiqui a un fitxer script.

A la figura 4.1, veiem un diagrama per entendre millor la funcionalitat de ScoreMan.

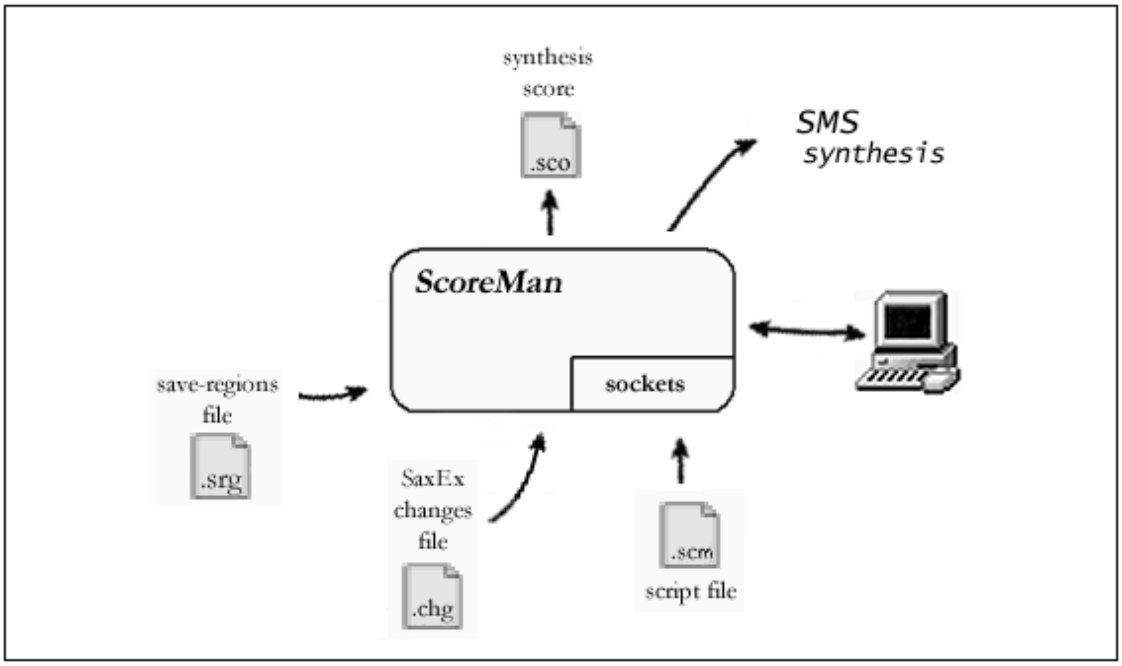


Figura 4.1: Esquema de la interfície de ScoreMan.

4.2- Fitxers de ScoreMan

ScoreMan treballa amb 4 tipus de fitxers:

- Fitxers *Save Regions*: tenen extensió *.srg i són els fitxers que contenen les regions d'un fitxer analitzat amb SMS (veure capítol 3 per explicació amb mes detall). Aquest fitxer conté informació sobre el número total de regions del fitxer i a continuació totes les regions, una darrera l'altre separades per salts de línia i cadascuna amb informació útil sobre ella com el número de regió, el tipus, el començament i durada (en segons), la freqüència mitja (*average pitch*) de la

regió, i el número de nota a la que pertany, 0 en el cas de ser una regió silenci. Veiem un exemple de fitxer SaveRegions resultant de la segmentació en regions de la primera frase del tema All Of Me (Allme_q1.snd) a la figura 4.12, notem que al gràfic es veu el fitxer en dos columnes.

```
59 Regions
0;4;0.000000;0.020317;0.000000;0
1;0;0.020317;0.023220;0.000000;0
2;1;0.043537;0.037732;0.000000;1
3;2;0.081270;0.505034;484.965210;1
4;3;0.586304;0.058050;0.000000;1
5;1;0.644354;0.043537;0.000000;2
6;2;0.687891;0.304762;363.142456;2
7;3;0.992653;0.060952;0.000000;2
8;1;1.053605;0.031927;0.000000;3
9;2;1.085533;1.915646;302.292145;3
10;0;3.001179;0.020317;0.000000;0
11;0;3.021497;0.467302;0.000000;0
12;1;3.488798;0.040635;0.000000;4
13;2;3.529433;0.336689;483.484039;4
14;3;3.866122;0.055147;0.000000;4
15;1;3.921270;0.029025;0.000000;5
16;2;3.950295;0.290249;541.014526;5
17;3;4.240544;0.046440;0.000000;5
18;1;4.286984;0.037732;0.000000;6
19;2;4.324717;0.272834;483.953888;6
20;3;4.597551;0.058050;0.000000;6
21;1;4.655601;0.031927;0.000000;7
22;2;4.687528;0.446984;456.366821;7
23;3;5.134512;0.046440;0.000000;7
24;1;5.180952;0.020317;0.000000;8
25;2;5.201270;0.339592;385.188660;8
26;3;5.540862;0.060952;0.000000;8
27;1;5.601814;0.026122;0.000000;9
28;2;5.627937;3.091156;304.493439;9
29;0;8.719093;0.020317;0.000000;0
30;0;8.739410;0.542766;0.000000;0
31;1;9.282177;0.043537;0.000000;10
32;2;9.325714;0.786576;408.775055;10
33;3;10.112290;0.052245;0.000000;10
34;1;10.164535;0.031927;0.000000;11
35;2;10.196463;0.200272;365.007751;11
36;3;10.396735;0.066757;0.000000;11
37;1;10.463492;0.029025;0.000000;12
38;2;10.492517;1.628299;302.715332;12
39;3;12.120816;0.026122;0.000000;12
40;1;12.146939;0.066757;0.000000;13
41;2;12.213696;0.458594;287.288330;13
42;3;12.672290;0.046440;0.000000;13
43;0;12.718730;0.020317;0.000000;0
44;1;12.739048;0.026122;0.000000;14
45;2;12.765170;0.304762;302.573853;14
46;3;13.069932;0.040635;0.000000;14
47;1;13.110567;0.040635;0.000000;15
48;2;13.151202;0.325079;395.211426;15
49;3;13.476281;0.034830;0.000000;15
50;1;13.511111;0.020317;0.000000;16
51;2;13.531429;0.325079;405.463715;16
52;3;13.856508;0.046440;0.000000;16
53;1;13.902948;0.031927;0.000000;17
54;2;13.934875;1.082630;363.457367;17
55;3;15.017506;0.043537;0.000000;17
56;1;15.061043;0.063855;0.000000;18
57;2;15.124898;3.018594;325.117035;18
58;3;18.143492;0.034830;0.000000;18
```

Figura 4.2: Exemple de fitxer SaveRegions (allme_q1A.srg).

- Fitxers *SaxEx Changes*: tenen extensió *.chg i contenen els canvis a fer sobre el fitxer inexpressiu de cara a fer la síntesi del nou so. Aquest fitxer conté, a les primeres línies, el fitxer *SaveRegions* on estan les regions del fitxer al que s'aplicaran els canvis, el nom del fitxer Score que es generarà, el nom del fitxer sms resultant de l'anàlisi del so inexpressiu i el nom del fitxer de so expressiu que es generarà amb la síntesi. A partir de les següents línies conté el número de nota a la qual es vol fer el canvi, el tipus de canvi i els paràmetres del canvi. De moment els canvis que suporta ScoreMan són canvis en l'amplitud de la nota

(*amp*) i en la durada (*stretch*), però es pot ampliar molt fàcilment en un futur. Podem veure un exemple de fitxer *SaxEx Changes* a la figura 4.3.

```
InputRegionsFile c:\Prog\sms\test\allme.srg
OutScoreFile c:\Prog\sms\test\allme2.sco
InputSMSFile c:\Prog\SMS\test\OutSms\aut-q1a2.sms
OutputExpFile c:\Prog\SMS\test\OutSnd\autA2-exp1.snd
1 amp 2
2 stretch 0.5
4 stretch 3
3 stretch 0.2
5 stretch 1.7
7 amp 2.2
10 amp 0.8
12 stretch 0.5
13 amp 0.7
13 stretch 0.8
15 amp 0.9
17 amp 0.6
```

Figura 4.3: Exemple de fitxer SaxEx Changes.

- Fitxers *Script*: tenen extensió *.scm i contenen ordres que interpreta el programa per tal de generar el fitxer de síntesi de forma automàtica o fins i tot cridar al sms per tal de fer la síntesi d'una manera automàtica. Tenen molta utilitat per fer la generació del fitxer score de síntesi remotament quan el programa es configura com a servidor. Veiem un exemple de fitxer script que genera un score i el sintetitza.

```
GenerateScore c:\mistyA1.chg
Synthesis 1
```

Figura 4.4: Exemple de fitxer Script

- Fitxers *Score*, tenen extensió *.sco i són els fitxers estàndards per representar paràmetres d'anàlisi i síntesi a SMS, en aquest cas ens interessa l'score de síntesi, que és el que haurem de generar mitjançant ScoreMan, per tal de després

generar el so expressiu amb síntesi SMS. Està compost de paràmetres que són constants durant tot el fitxer de so, i paràmetres envolupants que són funcions que poden variar al llarg del fitxer. Veiem un exemple de Score de síntesi.

```
InputSmsFile c:\Prog\SMS\test\OutSms\aut-q1a2.sms
OutputSoundFile c:\Prog\SMS\test\OutSnd\autA2-exp1.snd
Mix 0
Type 1
SamplingRate 44100
WindowSize 256
TimeStretch 0.0020 1 0.0021 0.5000 0.0151 0.5000 0.0152 1
TimeStretch 0.0179 1 0.0180 2.0000 0.0259 2.0000 0.0260 1
Amp 0.0938 1 0.0939 0.8000 0.1007 0.8000 0.1008 1
TimeStretch 0.1039 1 0.1040 1.7000 0.1115 1.7000 0.1116 1
Amp 0.1244 1 0.1245 1.2000 0.1339 1.2000 0.1340 1
Amp 0.2465 1 0.2466 1.3000 0.2654 1.3000 0.2655 1
TimeStretch 0.2687 1 0.2688 0.2000 0.2738 0.2000 0.2739 1
TimeStretch 0.2765 1 0.2766 0.5000 0.3188 0.5000 0.3189 1
```

Figura 4.5: Exemple de fitxer Score de síntesi generat per ScoreMan

4.3- Representació interna i jerarquia de classes

El programa implementa una sèrie de classes en C++ amb unes interrelacions entre elles. Veiem primer a la figura 4.6 la llista de totes les classes existents al programa, i ens centrarem desprès en que fa cadascuna i com s'interrelacionen entre elles. Veiem que les classes amb nom acabat en Doc corresponen a classes document encarregades de guardar i tractar les dades llegides dels fitxers associats i les classes acabades en View corresponen a classes vista que s'encarreguen de visualitzar les dades guardades al document associat.

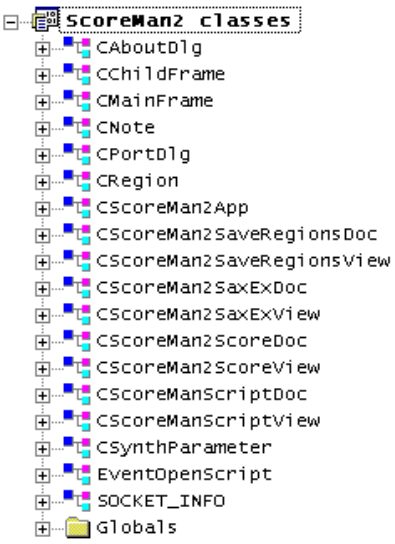


Figura 4.6: Classes existents a ScoreMan

La representació d'una nota ve definida per la classe CNote, que a la vegada té tres variables membre (mAttack, mSteadyState y mRelease) que són instàncies de la classe CRegion. Veiem a la figura 4.7 aquestes dues classes amb les seves variables i funcions membre, per entendre millor la interrelació entre elles.

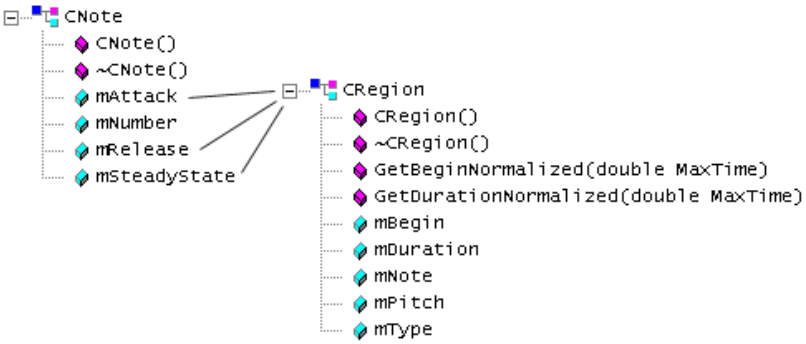


Figura 4.7: Les classes CNote i CRegion.

També tenim una classe anomenada CSynthParameter que té la funció de guardar en un format intern propi del programa, els paràmetres de síntesi referents als canvis indicats al fitxer SaxEx, per tal de després fer la conversió a format SMS. Aquesta classe té com a variables membre, el tipus del paràmetre, el seu valor i entre quins dos instants de temps s'ha d'aplicar.

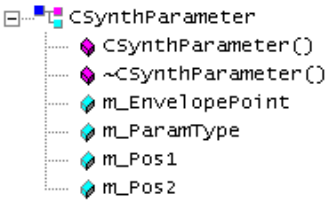


Figura 4.8 : La classe CsynthParameter.

A part d'aquestes classes que hem comentat que tenen una funcionalitat més d'estructura per a guardar dades que de processament, per a cada tipus de fitxer diferent tenim associada una classe document que és la que s'encarrega de llegir les dades del fitxer de disc i posar-les en aquestes estructures, per tal de fer més fàcil la organització, visualització i conversió posterior. Per fer la visualització de cada un dels fitxers, cada classe document té associada la seva vista pròpia, que mostra per pantalla les dades en el format que convingui en cada cas. Al carregar un fitxer de canvis SaxEx, algunes dades es carreguen a la classe genèrica de l'aplicació, per tant, no serà possible carregar dos fitxers de canvis simultàniament, només serà possible en el cas de que un dels fitxers de canvis es carregui mitjançant un script, ja que en aquest cas, les dades de l'script es carreguen a la classe CscoreManScriptDoc, que és independent de totes les altres, i guarda internament totes les dades necessàries per al seu processament.

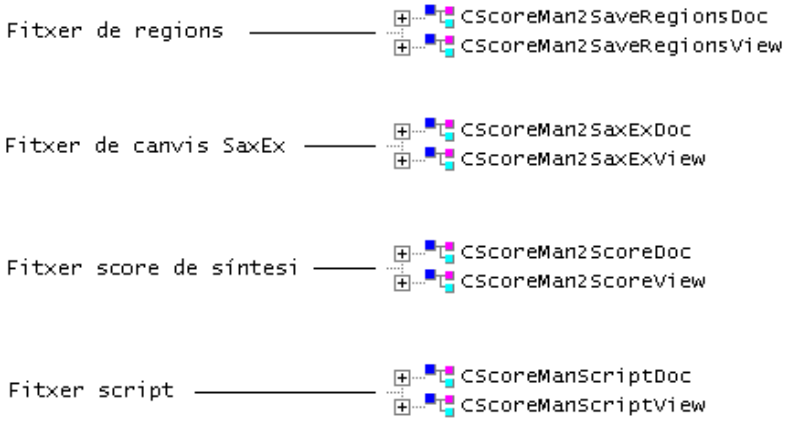


Figura 4.9: Relacions entre fitxers i classes a ScoreMan.

Cada classe document té una funció membre associada anomenada Serialize que és l'encarregada de controlar l'input i l'output de les classes, per tal de carregar les dades del fitxer de disc a les structures internes de ScoreMan i per tal de gravar a disc,

en cas de que l'usuari ho demani, els diferents resultats obtinguts durant el programa. Per tant, cada cop que es carrega un fitxer al programa, es crida a la funció serialize de la classe document associada al tipus de fitxer que es vol carregar. Veiem a la figura 4.9 per a cada tipus de fitxer les classes associades:

Veiem que succeeix internament quan es carrega un fitxer de canvis SaxEx i es crida a la funció de generar el score de síntesi i a la funció de síntesi SMS. Agafem aquest exemple, perquè així engloba també la càrrega d'un fitxer de regions, que és fa automàticament, llegint el path indicat al paràmetre *InputRegionsFile* del fitxer de canvis carregat.

Al carregar el fitxer de canvis, el primer pas que es fa és crear un array d'objectes del tipus *CSynthParameter* on es guardaran els diferents paràmetres de síntesi que es llegiran del fitxer, per després convertir-los al format SMS. Aquest array es troba a la classe *CScoreMan2App* (veure figura 4.10) amb el nom *m_pSPArray*. El següent pas és llegir els diferents paràmetres del fitxer de canvis i guardar-los també a la classe *CScoreMan2App* amb els noms següents: *m_SaveRegionsFileName*, *m_ScoreFileName*, *m_InputSMSFile* i *m_OutputSoundFile*.

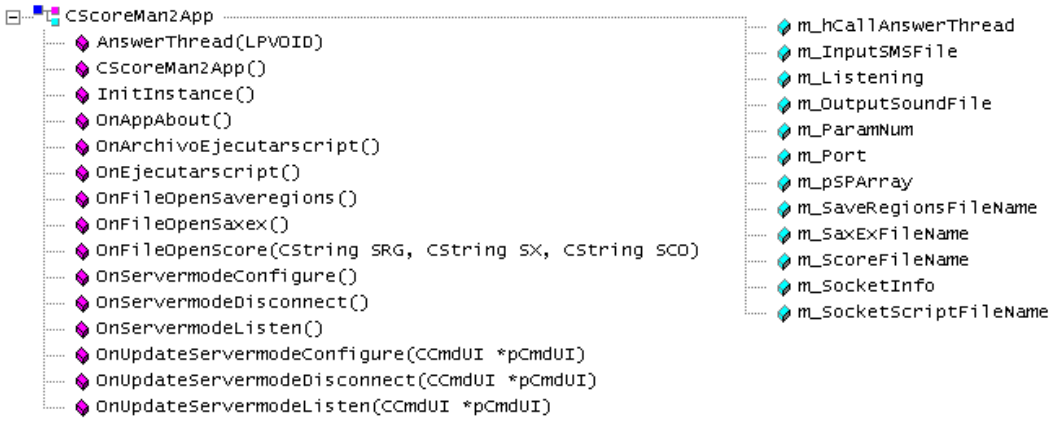


Figura 4.10: Estructura de la classe principal de l'aplicació ScoreMan.

Un cop llegits aquests paràmetres, fent servir el path llegit del paràmetre *m_SaveRegionsFileName*, carreguem aquest fitxer de regions, guardant totes les regions del fitxer en una estructura interna de l'objecte *CScoreMan2SaveRegionsDoc* (veure figura 4.11) anomenada *m_pRegionsArray*, que és un array format d'objectes del tipus *CRegion* que hem vist abans. També es carreguen unes variables ínternes de la classe com són el núnero total de notes, regions i temps màxim del fitxer i es crea un array anomenat *m_pNotesArray*, on es graven totes les notes existents a les regions del fitxer

com a objectes *CNote*, cada un amb les seves regions atac, steady state o release segons correspongui.

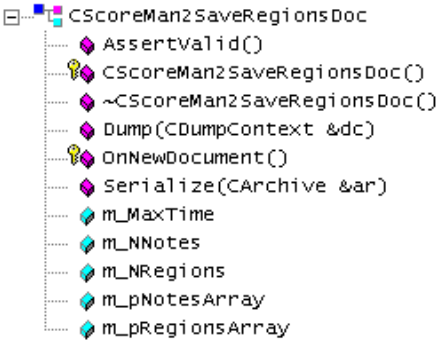


Figura 4.11: Estructura de la classe que guarda les regions i notes del fitxer *.srg.

El següent pas, un cop ja s'han gravat les regions i notes a memòria, consisteix en gravar també a l'array creat abans (*m_pSPArray*) tots els paràmetres de síntesi que es troben al fitxer de canvis SaxEx. Llavors al cridar a la funció de la classe *CScoreMan2SaxExDoc* (veure figura 4.12) encarregada de generar l'score de síntesi, anomenada *OnGenScore()*, s'obrirà un nou document del tipus *CScoreMan2ScoreDoc* (veure figura 4.13), i es convertiran els paràmetres de síntesi a un format que entén SMS, per tal de que la classe document del score pugui gravar el fitxer score a disc.

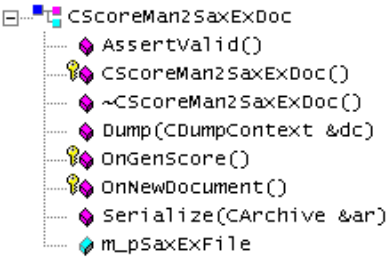


Figura 4.12: Estructura de la classe document associada als fitxers de canvis *.chg.

Finalment, si volem sintetitzar fent servir el nou fitxer score creat, podem cridar a la funció *OnSynthSmssynthesis()* de la classe *CScoreMan2ScoreDoc* que fa l'execució remota del programa SMS per tal de sintetitzar el nou so expressiu.

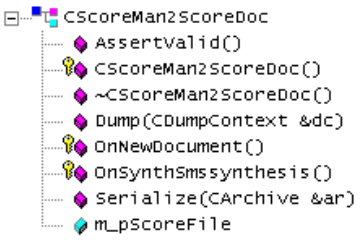


Figura 4.13: Estructura de la classe document associada als fitxers score *.sco.

Per altra banda, en el cas de l'execució d'un fitxer script, tot el procés es fa dintre d'una sola classe anomenada *CScoreManScriptDoc* (veure figura 4.14), de forma que les dades del procés que executa el script són privades a aquesta classe, permetent que l'execució del script es pugui integrar en un thread diferent al de l'aplicació principal, de forma que no interfereixi amb el procés de l'usuari que està a la consola interactuant amb els menús del programa, visualitzant fitxers, generant scores o sintetitzant sons expressius. Això és realment molt important, quan el programa està configurat en mode servidor, ja que d'aquesta manera, podem permetre a un usuari utilitzar el programa des de la consola, encara que altres usuaris estiguin executant scripts remotament.

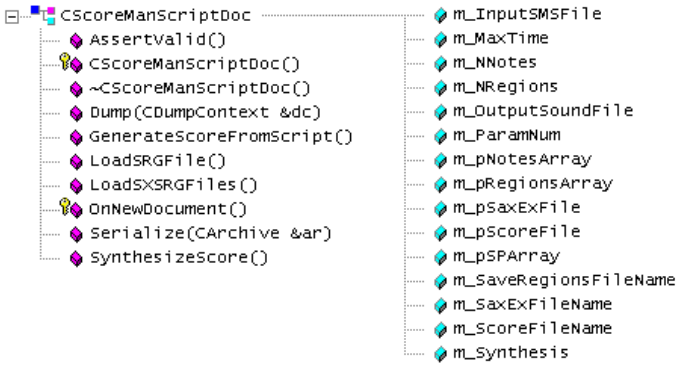


Figura 4.14: Estructura de la classe document associada als fitxers script *.scm.

4.4- Funcionament

L'objectiu final de ScoreMan és generar un Score de síntesi i sintetitzar un nou so a partir del resultat de l'anàlisi del fitxer original amb SMS i del fitxer de canvis produït

per SaxEx. Veiem com es realitza aquest procés a dintre de ScoreMan. Independentment de que ha de fer l'usuari per obtenir els resultats, que ja es comenta en un altre apartat, veurem com es genera internament el fitxer score de síntesi.

Com ja s'ha dit al principi del capítol 3, molts dels paràmetres de SMS, tant d'anàlisi com de síntesi són envolupants, és a dir, són una funció que pot variar al llarg del temps, de fet els paràmetres que s'han d'incloure a l'score per tal de generar el so expressiu faran referència a una nota del fitxer, per tant han de ser envolupants perquè només tenen sentit durant el temps que dura la nota a la qual s'aplica el canvi.

De moment s'accepten dos tipus de transformacions com són canvis en amplitud d'una determinada nota o canvis en la durada de la nota sense modificar la seva freqüència, de cara a afegir expressivitat a una melodia gravada. Així doncs ens podem trobar, per exemple, que la sortida de SaxEx demana que la nota 5 es redueixi a la meitat de la seva durada, o que la nota 7 s'amplifiqui al doble.

4.4.1- Canvis d'amplitud

Veiem un exemple perquè quedi clar el procediment seguit per modificar l'amplitud d'una nota. Suposem que tenim una sentencia SaxEx que ens demana que amplifiquem la nota 4 al doble del seu valor de la següent manera:

```
4 amp 2
```

I que el fitxer de regions és el següent:

```
59 Regions
0;4;0.000000;0.020317;0.000000;0
1;0;0.020317;0.023220;0.000000;0
2;1;0.043537;0.037732;0.000000;1
3;2;0.081270;0.505034;484.965210;1
4;3;0.586304;0.058050;0.000000;1
5;1;0.644354;0.043537;0.000000;2
6;2;0.687891;0.304762;363.142456;2
```

```
7;3;0.992653;0.060952;0.000000;2
8;1;1.053605;0.031927;0.000000;3
9;2;1.085533;1.915646;302.292145;3
10;0;3.001179;0.020317;0.000000;0
11;0;3.021497;0.467302;0.000000;0
12;1;3.488798;0.040635;0.000000;4
13;2;3.529433;0.336689;483.484039;4
14;3;3.866122;0.055147;0.000000;4
15;1;3.921270;0.029025;0.000000;5
16;2;3.950295;0.290249;541.014526;5
17;3;4.240544;0.046440;0.000000;5
18;1;4.286984;0.037732;0.000000;6
19;2;4.324717;0.272834;483.953888;6
20;3;4.597551;0.058050;0.000000;6
...
58;3;18.143492;0.034830;0.000000;18
```

Ens hem de centrar a les regions 12, 13 i 14 del fitxer que corresponen a l'atac, part transitòria i final respectivament de la nota 4 que és la que hem d'amplificar. Per fer el canvi en l'amplitud de la nota farem servir un paràmetre de síntesi existent a SMS anomenat *Amp*. Aplicarem el canvi d'amplitud només sobre la part transitòria de la nota, per aconseguir un efecte de naturalitat a la síntesi, ja que si amplifiquem o atenuem massa l'atac o el *release* d'una nota, ens trobarem amb un cas que no es produeix normalment a la realitat i l'efecte no seria gens agradable per l'oïda. Llavors al cas de l'exemple aplicaríem l'amplificació sobre la regió 13 que comença al segon 3.529433, dura 0.336689 segons i té una freqüència mitjana de 483.484039 Hz.

Gràficament podem representar aquesta nota aïllada de la següent manera:

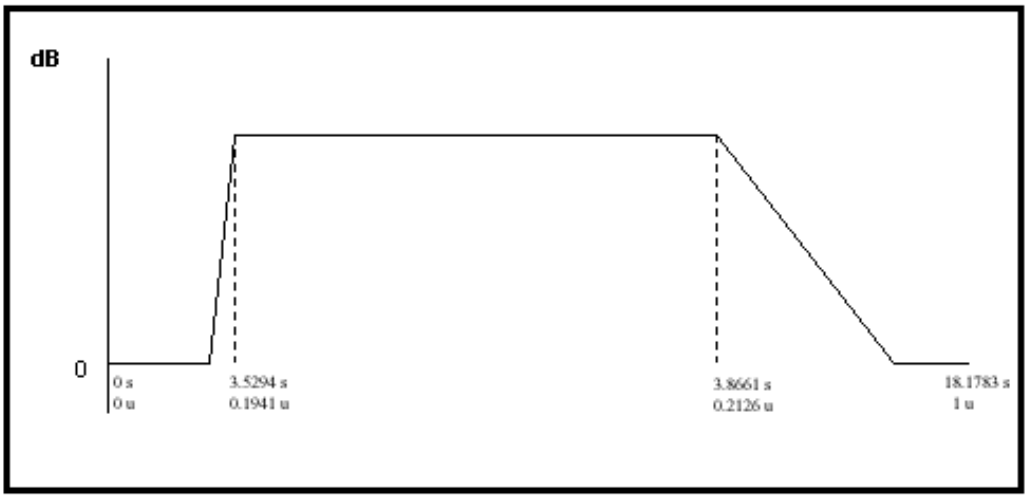


Figura 4.15: Temps absoluts i relatius de la regió 13 (nota 4) de l'exemple anterior.

Els paràmetres envolupants tant d'anàlisi com de síntesi a SMS treballen amb unitats de temps relatives a la durada total del fitxer, essent així el temps 0 l'inici del fitxer i el temps 1 el final del fitxer, d'aquesta manera haurem de fer la següent conversió de cara a obtenir els instants de temps on comença i acaba la regió a la qual volem aplicar el canvi en l'amplitud:

```
    Temps relatiu SMS       = Temps Absolut / Durada total del fitxer
(normalitzat entre 0 i 1)
```

En el cas de l'exemple, la durada total del fitxer és de 18.178322 segons, llavors els temps relatius de la regió 13 quedaran:

```
Temps relatiu on comença = 3.529433 s / 18.178322 s = 0.194156 unitats

Durada relativa  = 0.336689 s / 18.178322 s = 0.018521 unitats
```

i el resultat final del fitxer score, per fer aquesta transformació d'amplitud quedarà de la següent manera:

```
Amp 0.1940 1 0.1941 2 0.2126 2 0.2127 1
```

Notem que els punts on l'amplitud es manté igual, són punts situats just abans i després del començament i final de la regió *steady state*, per tant ni l'atac ni el *release* es veuran afectats pel canvi d'amplitud.

4.4.2- Canvis de durada (time stretching)

Igual que hem fet al punt anterior, veiem un exemple perquè quedi clar el procediment seguit per modificar la durada d'una nota. Hem de tenir en compte que encara que la durada de la nota canviï, la freqüència de la nota no és veurà afectada, ni

tampoc es farà un tall o un allargament sense respectar les possibles variacions que pugui haver en l'amplitud o *pitch* de la nota, el procés de *time stretching* és molt més elaborat. Suposem que tenim una sentencia *SaxEx* que ens demana que allarguem la nota 2 a 1,2 vegades la durada actual de la següent manera:

```
2 stretch 1.2
```

I que el fitxer de regions és el següent:

```
59 Regions
0;4;0.000000;0.020317;0.000000;0
1;0;0.020317;0.023220;0.000000;0
2;1;0.043537;0.037732;0.000000;1
3;2;0.081270;0.505034;484.965210;1
4;3;0.586304;0.058050;0.000000;1
5;1;0.644354;0.043537;0.000000;2
6;2;0.687891;0.304762;363.142456;2
7;3;0.992653;0.060952;0.000000;2
8;1;1.053605;0.031927;0.000000;3
9;2;1.085533;1.915646;302.292145;3
10;0;3.001179;0.020317;0.000000;0
11;0;3.021497;0.467302;0.000000;0
12;1;3.488798;0.040635;0.000000;4
13;2;3.529433;0.336689;483.484039;4
14;3;3.866122;0.055147;0.000000;4
...
58;3;18.143492;0.034830;0.000000;18
```

Ens hem de centrar a les regions 5, 6 i 7 del fitxer que corresponen a l'atac, part transitòria i final respectivament de la nota 2 que és la que hem d'amplificar. Per fer l'allargament o compressió de la durada de la nota farem servir un paràmetre de síntesi existent a SMS anomenat *TimeStretch* . Aplicarem també el *TimeStretching*, igual que el canvi d'amplitud, només sobre la part transitòria de la nota, per aconseguir un efecte de naturalitat a la síntesi, ja que si acurtem o estirem massa l'atac o el *release* d'una nota, ens trobaríem amb casos indesitjables com atacs massa llargs o massa curts que no serien gens agradables per l'oïda. Llavors al cas de l'exemple aplicaríem el *Stretching* sobre la regió 6 que comença al segon 0.687891, dura 0.304762 segons i té una freqüència mitjana de 363.142456 Hz.

Gràficament podem representar aquesta nota aïllada de la següent manera:

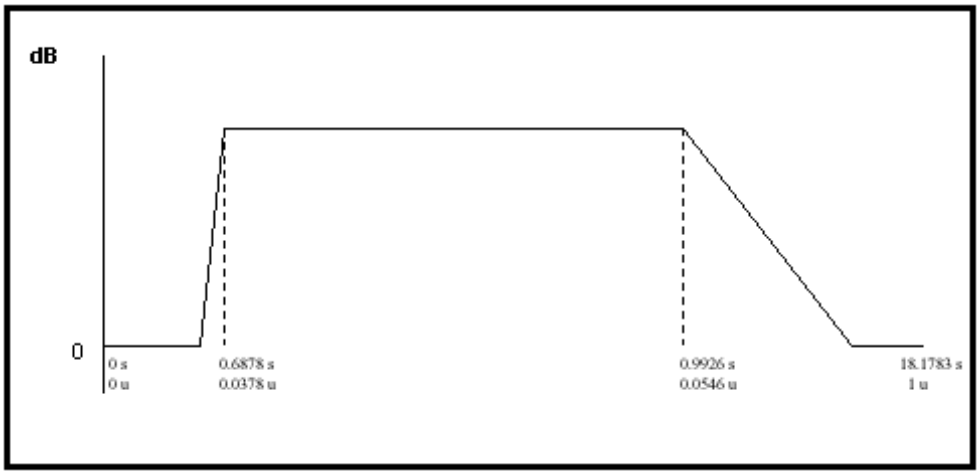


Figura 4.16: Temps absoluts i relatius de la regió 6 (nota 2) de l'exemple anterior.

En aquest cas, la durada total del fitxer també és de 18.178322 segons, i els temps relatius de la regió 6 quedaran:

```
Temps relatiu on comença = 0.687891 s / 18.178322 s = 0.037841 unitats

Durada relativa    = 0.304762 s / 18.178322 s = 0.016765 unitats
```

i el resultat final del fitxer *score*, per fer aquesta transformació d'amplitud quedarà de la següent manera:

```
TimeStretch  0.0377  1  0.0378  1.2  0.0546  1.2  0.0547  1
```

Notem que els punts on el paràmetre envolupant (TimeStretch) val 1, marquen els límits entre els quals es fa la compressió o allargament de la durada de la nota, que estan situats just abans i després del començament i final de la regió *steady state*, per tant ni l'atac ni el *release* es veuran afectats per la transformació.

4.5- El GUI

ScoreMan està dotat d'una completa interfície gràfica amb menús i vistes d'arxius desenvolupada utilitzant les *MFC* i *Visual Studio 6.0* per *Windows 98*. S'ha fet servir la filosofia de *Visual C++* document/vista, per tal de tenir cada tipus de fitxer en una classe document diferent i que cada classe tingui associada una vista, depenent del tipus de fitxer associat, per així poder representar les dades dels fitxers d'una manera més llegible i entenedora pensant en l'usuari final.

L'aspecte del menú principal del programa és el següent

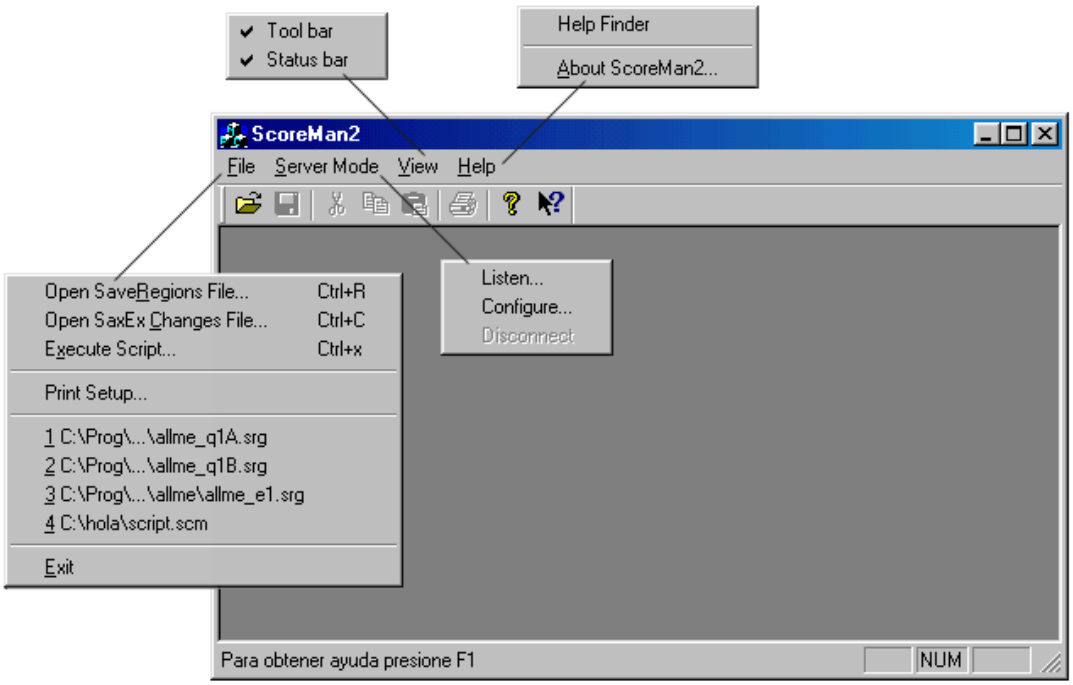


Figura 4.17: Aspecte visual del menú principal de ScoreMan

A part dels menús *View* i *Help* que no tenen gaire rellevància per l'objectiu d'aquest projecte, trobem dos menús més que són: *File* i *Server Mode*. Del menú *Server Mode*, que és el que permet configurar el programa com a servidor i acceptar peticions remotes a través d'un socket en parlem més endavant, i el menú *File*, permet carregar tres tipus de fitxers:

- Fitxers SaveRegions
- Fitxers SaxEx Changes
- Scripts d'execució

Veiem per separat el procediment de càrrega d'aquests fitxers i la interfície visual i sistema de menús que ScoreMan ens proporciona per a cada cas.

4.5.1- Visualització de fitxers SaveRegions

L'opció de menú "*Open SaveRegions File...*" igual que totes les altres del tipus "*Open...*" obre un quadre de diàleg que ens permet navegar per les carpetes del disc per seleccionar el fitxer que volem obrir (d'extensió *.srg) per tal de visualitzar-lo per pantalla. Si mirem la figura 4.18, podem veure el resultat de carregar 4 fitxers de

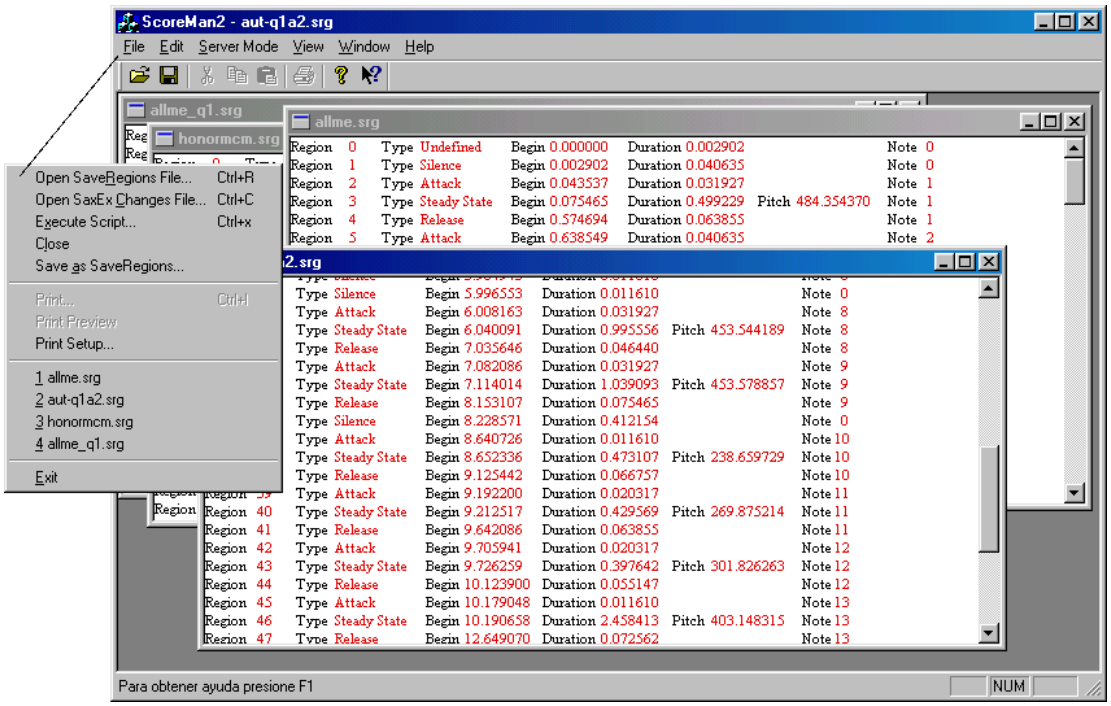


Figura 4.18: Visualització de 4 fitxers de regions (*.srg) simultàniament.

regions i visualitzar-los a la vegada, si ens fixem, la representació gràfica dels fitxers, és molt més entenedora que la versió guardada al fitxer, permetent així a l'usuari tenir clar, sobre quines regions o notes es faran els canvis, saber el tipus i número de nota de cada regió i en cas de ser una regió estable, saber la seva freqüència mitjana.

Notem també que els menús del programa són contextuals, per tant, depenent del fitxer que tinguem carregat, podrem fer unes accions o unes altres. En aquest cas, si tenim carregat un fitxer de regions, el menú *File* ens permetrà obrir un altre fitxer de regions, obrir un fitxer de canvis, executar un script, tancar el fitxer obert o bé gravar el fitxer amb un altre nom.

També hem de tenir present que encara que tinguem algun fitxer obert, podem posar el programa en mode servidor per acceptar peticions remotes, sense que aquestes interfereixin en el bon funcionament del programa. De cada fitxer nou que s'obre, el programa crea una instància nova de la classe que guarda el document i cada classe document té associada una classe vista que s'encarrega de visualitzar les dades per pantalla en el format que es veu a la figura 4.18.

4.5.2- Visualització de fitxers de canvis SaxEx

El procediment de càrrega d'un fitxer de canvis SaxEx és molt semblant al d'un fitxer *SaveRegions*, un cop seleccionat el fitxer desitjat, que tindrà extensió *.chg, s'obre el fitxer i el programa automàticament carrega també el fitxer de regions associat i visualitza per pantalla tots dos, obtenint un resultat semblant al de la figura 4.19.

Un cop carregats els fitxers, als menús apareix una nova opció que fa referència a la possibilitat de generar un fitxer *score* de síntesi SMS que inclourà els canvis especificats al fitxer carregat en un format que entendrà SMS. Altres opcions que apareixen als menús són tancar qualsevol dels fitxers oberts, obrir un altre fitxer diferent o gravar a disc els fitxers oberts.

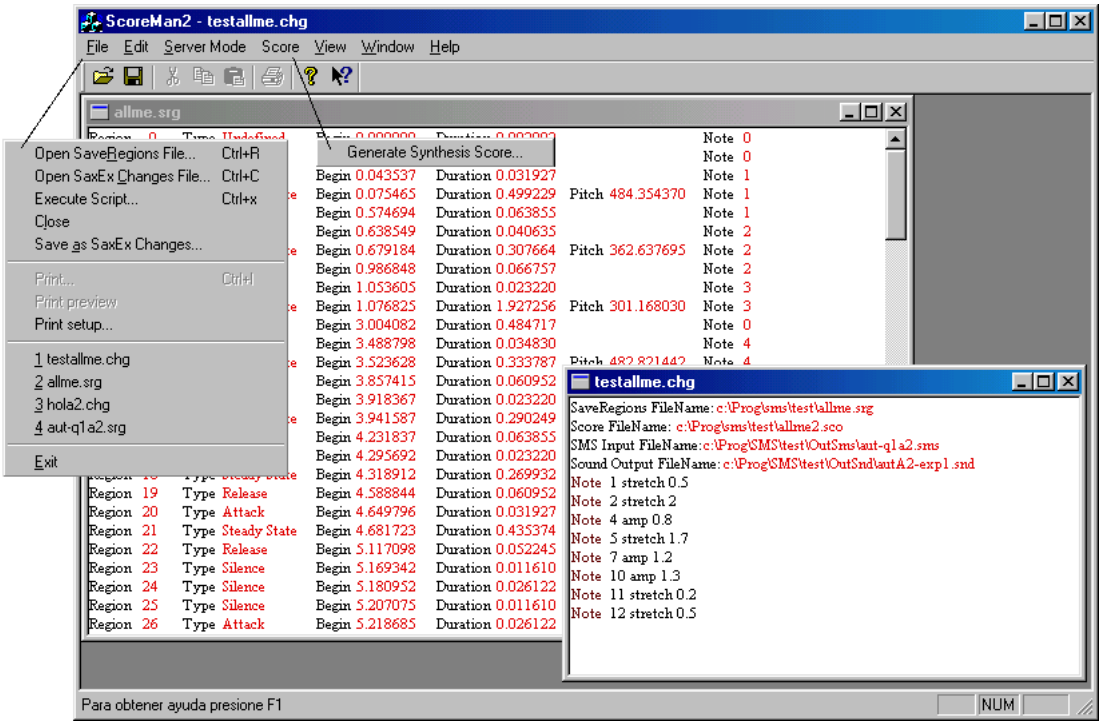


Figura 4.19: Resultat de la càrrega d'un fitxer de canvis de SaxEx (*.chg)

Si generem el fitxer score de síntesi, ens apareixerà per pantalla, tal i com es veu a la figura 4.20, el resultat de la generació automàtica del fitxer a partir dels canvis i paràmetres existents al fitxer *.chg i s'afegeixen també uns paràmetres de síntesi SMS que es troben en un fitxer de configuració anomenat scoreman.ini, que té un format com aquest:

```
[ScoreParams]
Mix 0
Type 1
SamplingRate 44100
WindowSize 256
```

D'aquesta manera, qualsevol usuari pot canviar al seu gust aquests paràmetres o afegir-ne altres, per adaptar-se al tipus de síntesi SMS desitjada, no entrarem en detalls del significat d'aquests paràmetres, ja que no és l'objectiu d'aquest projecte. Aquests paràmetres s'han posat en un fitxer extern, perquè no té cap sentit controlar-los des del fitxer de canvis SaxEx i perquè en molts casos es faran servir els que hi ha per defecte.

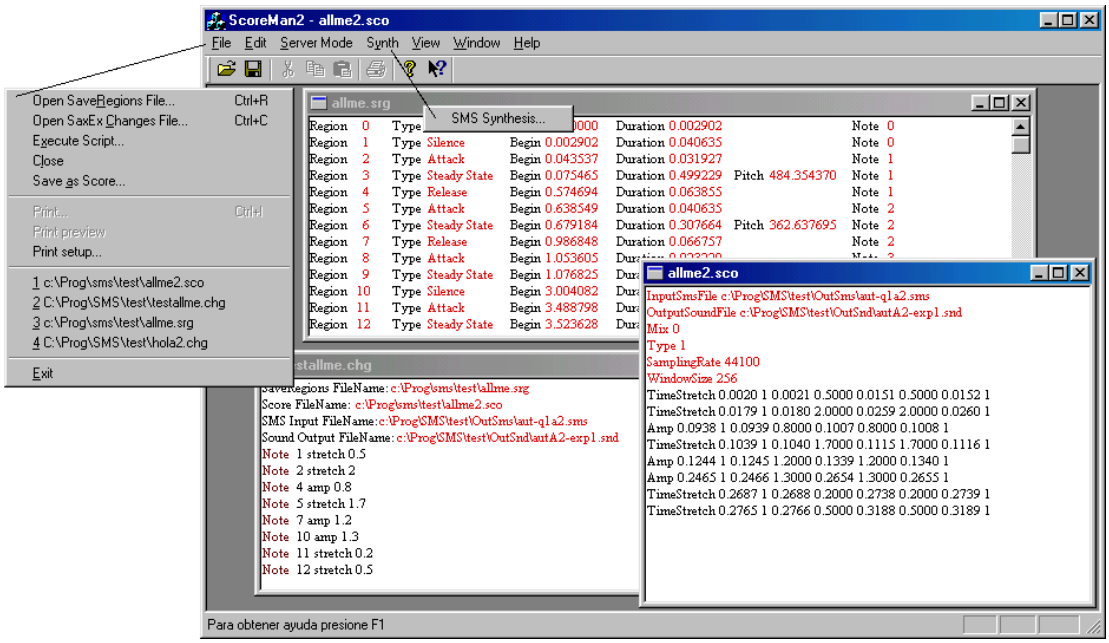


Figura 4.20: Score generat a partir del fitxer de canvis de la figura 4.19.

Un cop s'ha generat el *score* de síntesi, una nova opció apareix al menú, que ens permet cridar a *SMS* per tal de sintetitzar el nou só expressiu fent servir el score que tenim a pantalla. Aquest procediment es fa mitjançant una crida remota (*WinExec*) al software *SMS* passant com a paràmetre el *score* generat, que conté informació sobre el fitxer *SMS* resultant de l'anàlisi prèvia, el nom del fitxer amb que es vol guardar el so expressiu i una sèrie de paràmetres de síntesi que entén *SMS* referents a transformacions sobre el so inexpressiu.

4.5.3- Execució de fitxers Script

La finalitat dels fitxers *script* és realitzar una sèrie de tasques d'una manera automàtica i ràpida, de manera que no ens haguem de preocupar d'anar buscant als menús la opció desitjada en cada moment. Tot i que s'ha dotat al programa d'una sèrie de *shortcuts* (tecles abreujades) per tal d'agilitzar algunes tasques com obrir fitxers, generar el *score*, sintetitzar el so o tenir una llista dels fitxers més recentment oberts, els

scripts són necessaris quan volem realitzar totes aquestes tasques en un sol pas i no necessitem visualitzar els fitxers, sinó obtenir un resultat.

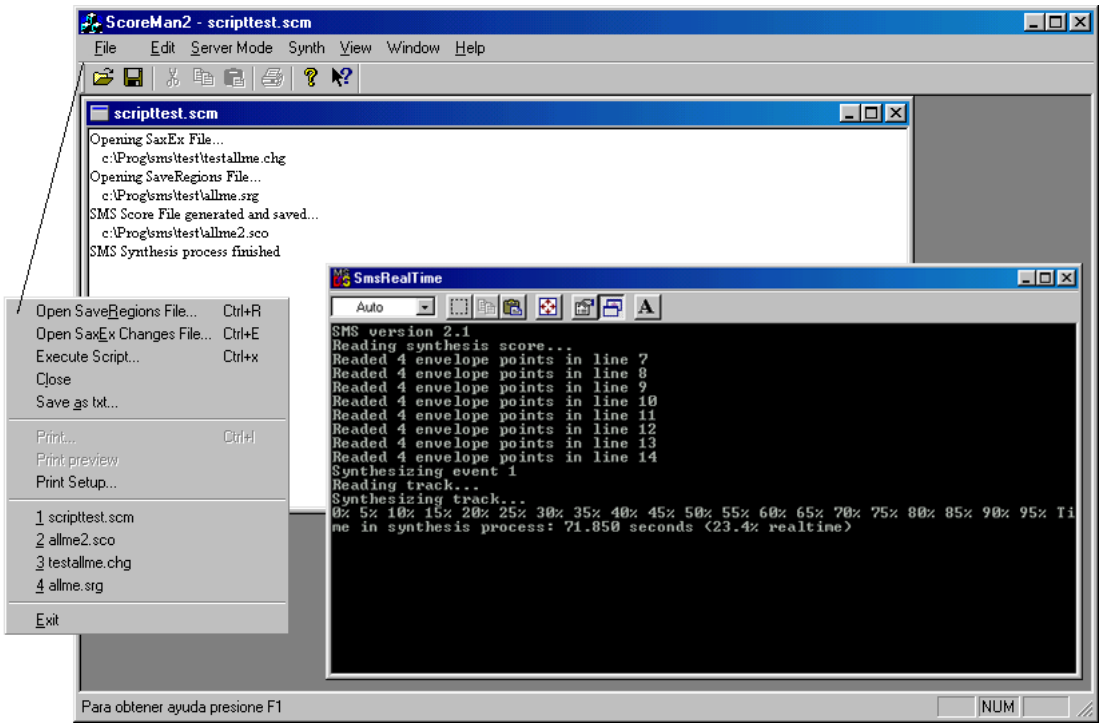


Figura 4.21: Resultat de l'execució d'un script amb crida de síntesi a SMS.

Els fitxers *script* tenen extensió *.scm i a la figura 4.21 podem veure el resultat d'executar el *script* següent:

```
GenerateScore c:\Prog\sms\test\testallme.chg
Synthesis 1
```

Generalment els *scripts* que accepta *ScoreMan* tindran aquestes dues línies, la primera, *GenerateScore* té com a paràmetre el path d'un fitxer de canvis *SaxEx* i farà que el programa generi un *score* de síntesi *SMS* a partir de la informació d'aquest fitxer. La segona línia, *Synthesis* accepta un booleà, i quan sigui 1 es farà la síntesi *SMS* i quan sigui 0 no es farà.

A la figura 4.21 podem veure que el programa va mostrant per pantalla els passos a mesura que els va fent, i també podem veure l'estat del procés de síntesi SMS a

la pantalla de consola que s'obre quan s'està sintetitzant el nou so. Un cop acabada l'execució de l'script, podem gravar a disc el fitxer de *log*, que conté tots els passos que s'han fet, així com el nom dels fitxers que s'han obert o generat. L'ús dels scripts és especialment útil quan es treballa amb el programa en *mode server*, que s'explica en el següent punt.

4.6- El model Client-Servidor

Com ja s'ha vist prèviament, el programa *ScoreMan* permet configurar-se com a servidor i acceptar peticions remotes d'execució d'un script per una xarxa amb el protocol TCP/IP. Això és fa mitjançant la opció del menú Server Mode, on tenim les següents opcions:

- L'opció *Listen* obre un port, crea un *socket*, vincula el *socket* al port i executa un *thread* que queda en espera de peticions d'execució d'scripts per part d'usuaris remots. El programa queda configurat com a server però és transparent per l'usuari que està a la consola on s'executa el programa, permetent així que cada nova petició s'executi en un *thread* diferent i que l'usuari pugui continuar visualitzant fitxers, fent conversions per generar *scores* de síntesi o fent crides remotes a SMS per sintetitzar nous fitxers. Una de les raons per les quals s'ha dotat d'aquesta funcionalitat a *ScoreMan* és el fet de que *SaxEx* està dissenyat per plataforma Macintosh amb el sistema *MacOS* i *ScoreMan* corre sobre PC amb *Windows 98/2000*, llavors d'aquesta manera, es pot tenir un PC com a servidor executant *ScoreMan*, i remotament anar enviant els resultats del raonament de *SaxEx* al PC, perquè ell faci la conversió al fitxer *score* de síntesi i faci la síntesi *SMS* a aquest mateix PC, ja que la versió que s'ha ampliat de *SMS* també està compilada sobre aquesta plataforma i així s'allibera part de la càrrega de processador i memòria de la plataforma Macintosh, podem veure en detall la implementació de la opció *listen* i la gestió del *thread* a la figura 4.24 .

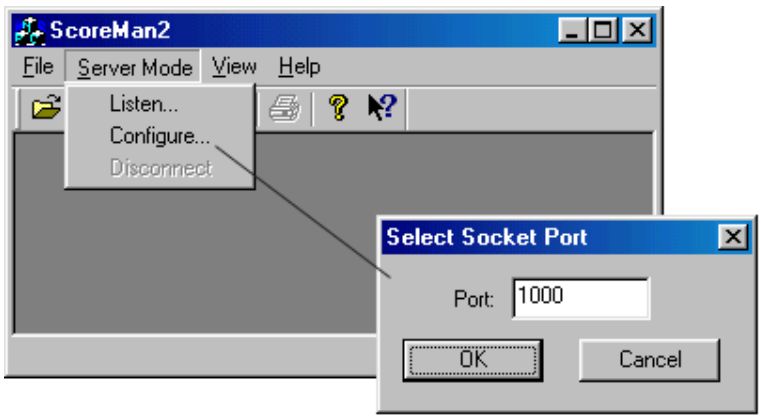


Figura 4.22: El menú Server Mode

- L'opció *Configure* permet canviar el port al qual es vincula el *socket* que escolta les peticions remotes (per defecte el port obert és el 1000).

- L'opció *Disconnect*, tanca el port i el *socket* que està escoltant i el programa segueix funcionant en mode normal.

La implementació de l'opció *listen* fa servir la classe CSocket definida a les MFCs, complementant-la amb una estructura (SOCKET_INFO) per guardar informació del socket com és l'estat del *socket* (bAbort, bConnected, bListening), els errors produïts (dwLastError) i el port al qual està vinculat (nPort) (veure figura 4.23). Per implementar els *threads* s'ha fet servir també una classe de les MFCs anomenada CWinThread.

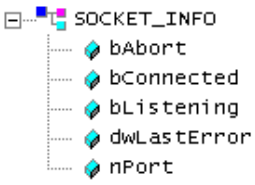


Figura 4.23: Estructura SOCKET_INFO

Veiem el codi de la funció que s'executa al prémer l'opció del menú *Listen* i el codi del *thread* que s'executa per escoltar peticions d'execució de *scripts* per part de clients. Si observem bé el codi veiem que El *thread* bàsicament realitza les següents operacions: Crea el *socket*, el posa en mode espera, escoltant pel port especificat, quan arriba una petició, estableix la connexió i espera a rebre dades, un cop es reben dades, que en el nostre cas s'espera el path d'un fitxer *script*, s'executen les comandes del fitxer, es retorna un missatge al client dient que tot és correcte i es torna a l'estat d'espera de noves peticions.

```
void CScoreMan2App::OnServermodeListen()
{
        CSocket ServerSocket;
        CSocket ClientSocket;
        BOOL bDone = FALSE;
        m_Listening = TRUE;

        if (m_SocketInfo.nPort == 0) m_SocketInfo.nPort = 1000;

        // Kick off the thread...
        CWinThread *lpThread = AfxBeginThread( CScoreMan2App::AnswerThread, (LPVOID) &m_SocketInfo );

        // Store the thread handle for later...
        m_hCallAnswerThread = lpThread->m_hThread;
}

UINT CScoreMan2App::AnswerThread( LPVOID lpInf )
{
        SOCKET_INFO *lpInfo = (SOCKET_INFO *) lpInf;
        CSocket ServerSocket;
        BOOL bDone = FALSE;
        CSocket ClientSocket;

        UINT nRetValue = 0;

Start:
        if( !ServerSocket.Create( lpInfo->nPort ) ) {  lpInfo->dwLastError = GetLastError();     // Record  the error.
                                                        nRetValue = 1;                            // Set our return value.
                                                       goto AnswerEnd;                            // Bail out.
                                                       }

        if( !ServerSocket.Listen( 1 ) ) {  lpInfo->dwLastError = GetLastError();              // Record the error.
                                            nRetValue = 2;                                     // Set our return value.
                                              goto AnswerEnd;                                  // Bail out.
                                           }

        if( !ServerSocket.Accept( ClientSocket ) )  {  lpInfo->dwLastError = GetLastError();      // Record the error.
                                                       nRetValue = 3;                             // Set our return value.
                                                       goto AnswerEnd;                            // Bail out.
                                                       }

        lpInfo->bConnected = TRUE;          // Set our connected flag to TRUE.
        MessageBeep( -1 );                  // Give an audio signal indicating that we're connected.

        while( !bDone ){
                char cbBuffer[1000];
                memset( cbBuffer, 0, sizeof( cbBuffer ) );

                // Read incoming data. This function blocks. It won't return until it has data or there's been a socket error.
                int nBytes = ClientSocket.Receive( cbBuffer, sizeof( cbBuffer ) );
```

```
                        // See if there was a problem. If so we need to bail out.
                        if( nBytes == 0 || nBytes == SOCKET_ERROR || lpInfo->bAbort )  bDone = TRUE;
                        else      // Otherwise, open the script.
                        {
                                CScoreMan2App* pApp = (CScoreMan2App*)AfxGetApp();
                                pApp->m_SocketScriptFileName.Format("%s",cbBuffer);
                                pApp->PostThreadMessage(ID_SOCKET_OPEN_SCRIPT,0,0);
                                ServerSocket.Close();
                                ClientSocket.Send("ok",2);        //return to client a message indicating that everything is ok
                                ClientSocket.Close();
                                goto Start;
                        }
            }

AnswerEnd:
            lpInfo->bListening = lpInfo->bConnected = FALSE;              // Reset our flags.

            CScoreMan2App* pApp = (CScoreMan2App*)AfxGetApp();
            pApp->m_Listening = FALSE;

            return( nRetValue );      // Return  to caller.
  }
```

Figura 4.24: Codi de la opció listen implementada amb CSockets i Threads.

Capítol 5: Proves

Per comprovar el correcte funcionament del projecte s'ha provat la segmentació en regions amb 5 temes estàndards de Jazz [12], cadascun interpretat amb diferents graus d'expressivitat des d'una versió totalment fidel a la partitura, fins a versions amb ús d'una gran quantitat de recursos expressius, en alguns casos molt exagerats, per veure els límits d'aplicació de l'algorisme.

Per entendre millor les taules que es mostren a continuació amb els resultats, veiem els tipus d'errors més comuns de segmentació que pot fer l'algorisme, amb una descripció breu sobre cada cas:

- *Vibrato*: aquests errors es produeixen quan l'algorisme interpreta com un canvi a una nova regió, una variació molt forta a l'amplitud i freqüència, provocada per un vibrato molt intens.

- *Glissando o Legato*: aquests errors es produeixen quan l'algorisme no interpreta com un canvi de nota, una transició suau, ja sigui deguda a un glissando molt exagerat o bé un legato molt suau.

- *Stacatto*: aquest error es produeix quan el so que s'està segmentant té una nota molt curta i aïllada, o bé quan canvia molt ràpid d'una nota a una altra i l'algorisme no detecta el canvi, sino que posa dues notes en una mateixa regió.

- *Altres*: qualsevol altre error no inclòs en els de dalt, com pot ser interpretar una regió steady state com a release o detectar un canvi de nota on no hi ha.

5.1 – Els casos no expressius

Veiem primer els resultats pels casos no expressius. A la taula veiem, a la primera columna, la frase de la melodia que s'ha analitzat, a les següents columnes el número d'errors que s'han comès de cada tipus i a la darrera columna, los notes que ha segmentat correctament l'algorisme respecte al total de notes de la frase i el càlcul del percentatge d'encert. A la darrera fila de la darrera columna veiem l'encert mig de tot el tema.

All Of Me

Allme_q0.snd: 4 frases amb 68 notes en total

Frase					Total
	vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	18/18=100%
B	-	-	-	-	16/16=100%
Abis	-	-	-	-	18/18=100%
C	-	-	-	-	16/16=100%
					68/68=100%

Allme_q1.snd: 4 frases amb 68 notes en total

Frase					Total
	vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	18/18=100%
B	-	-	-	-	16/16=100%
Abis	-	-	-	-	18/18=100%
C	-	-	-	-	16/16=100%
					68/68=100%

Als dos casos inexpressius analitzats del tema All of Me, veiem que els resultats són molt bons, l'algorisme segmenta correctament totes les notes de la melodia.

Autumn Leaves

Autum_q0.snd: 4 frases amb 68 notes en total

Frase					Total
	vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	17/17=100%
Abis	-	-	-	-	17/17=100%
B	-	-	-	-	17/17=100%
C	-	-	-	-	17/17=100%
					68/68=100%

Autum_q1.snd: 4 frases amb 68 notes en total

Frase					Total
	vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	17/17=100%
Abis	-	-	-	-	17/17=100%
B	-	-	-	-	17/17=100%
C	-	-	-	-	17/17=100%
					68/68=100%

Pel cas de la segmentació en regions del tema Autumn Leaves, veiem també que els resultats són totalment satisfactoris, amb un 100% d'encert.

My One and Only Love

Love_q1.snd: 4 frases amb 138 notes en total

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	2	33/35=94.3%
B	-	-	-	1	34/35=97.1%
C	-	-	-	-	33/33=100%
D	-	-	-	-	35/35=100%
					135/138=97.8%

En aquest cas veiem que les frases C i D s'han segmentat correctament, però a les dues primeres apareixen tres errors que corresponen a una regió *steady state* detectada com a *release*, degut a que l'amplitud de la nota és lleugerament decreixent durant tota la regió, i regions mal detectades per problemes de continuitat de pitch a la fase d'anàlisi durant una regió estable que porten a segmentar en dues notes, una de sola.

Misty

Misty_q1.snd: 4 frases amb 152 notes en total

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	39/39=100%
B	-	-	-	-	39/39=100%
C	-	-	-	-	35/35=100%
D	-	-	-	-	39/39=100%
					152/152=100%

Aquest cas ha estat segmentat correctament.

How High The Moon

Honormcm.au: 1 frase amb 18 notes

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	18/18=100%
					18/18=100%

Hownorm.au: 1 frase amb 18 notes

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	18/18=100%
					18/18=100%

Aquests casos també han estat segmentats sense cap problema.

5.2 – Els casos expressius

Ara veiem els resultats pels casos expressius. Veurem que els resultats no són tan bons com els casos no expressius, però la majoria dels errors són deguts a un vibrato molt intens, que podria ser extret en una fase d'anàlisi prèvia a la segmentació, i milloraria considerablement els resultats. El format de la taula és el mateix que l'explicat pels casos no expressius.

All Of Me

Allme_e1.snd: 4 frases amb 68 notes en total

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	18/18=100%
B	1	-	-	-	15/16=93.7%
Abis	2	-	1	-	15/18=83.3%
C	2	-	-	-	14/16=87.5%
					62/68=91.2%

Allme_e2.snd: 4 frases amb 68 notes en total

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	1	-	1	-	16/18=88.9%
B	1	1	-	1	13/16=81.2%
Abis	-	1	1	-	16/18=88.9%
C	1	-	-	-	15/16=93.7%
					60/68=88.2%

Allme_e3.snd: 4 frases amb 68 notes en total

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	18/18=100%
B	1	1	-	1	13/16=81.2%
Abis	1	-	-	-	17/18=94.4%
C	-	-	-	1	15/16=93.7%
					63/68=92.6%

Veiem que els errors comesos són majoritàriament per l'ús exagerat del *vibrato*, que esdevé en una decisió de canvi de nota quan hi ha un augment considerable en

l'amplitud i la freqüència. Altre error apareix quan es fan canvis massa ràpids entre notes amb intervals musicals petits, interpretant llavors dues notes com una de sola perquè no es detecta el canvi.

Autumn Leaves

Autum_e1.snd: 4 frases amb 68 notes en total

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	1	16/17=94.1%
Abis	3	-	1	-	13/17=76.5%
B	1	-	-	-	16/17=94.1%
C	1	2	-	-	14/17=82.3%
					59/68=86.8%

Tornen a produir-se errors deguts al vibrato, i en casos aïllats, trobem algun legatto molt suau no detectat com a tal i en algun cas en que hi ha canvis molt ràpids i notes molt curtes a la melodia, la detecció també falla.

My One and Only Love

Love_e1.snd: 4 frases amb 68 notes en total

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	1	-	-	34/35=97.1%
B	3	1	-	1	30/35=85.7%
C	-	3	-	-	30/33=90.9%
D	2	2	-	1	30/35=85.7%
					124/138=89.8%

Aquesta interpretació, a més dels vibratos exagerats, presents en quasi totes les interpretacions, trobem casos en que l'intèrpret fa servir el recurs expressiu glissando, que porta a l'algorisme a segmentar el glissando com a dues notes, pel canvi d'amplitud i de freqüència que es produeix. Altre problema que apareix novament és que l'amplitud d'una nota és lleugerament decreixent durant tota una regió estable, detectant-la erròniament com a release.

Misty

Misty_e1.snd: 4 frases amb 152 notes en total

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	2	-	-	1	36/39=92.3%
B	1	-	-	-	38/39=97.4%
C	1	2	-	-	32/35=91.4%
D	3	-	-	-	36/39=92.3%
					142/152=93.4%

Tornen a ser errors produïts principalment per un vibrato exagerat i algún glissando.

How High The Moon

Hocupom.au: 1 frase amb 18 notes

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	18/18=100%
					18/18=100%

Hobrillm.au: 1 frase amb 18 notes

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	1	-	-	17/18=94.4%
					17/18=94.4%

Howls.au: 1 frase amb 18 notes

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	1	1	-	1	15/18=83.3%
					15/18=83.3%

Howet.au: 1 frase amb 18 notes

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	1	1	-	16/18=88.9%
					16/18=88.9%

Hopes.au: 1 frase amb 18 notes

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	-	-	-	-	18/18=100%
					18/18=100%

Howcg.au: 1 frase amb 18 notes

Frase					Total
	Vibrato	legato/glissando	stacatto	altres	
A	2	1	-	-	15/18=83.3%
					15/18=83.3%

En aquests casos la segmentació funciona força bé, detectant en alguns casos totes les regions correctament, només fallant en casos aïllats pels problemes més comuns com són el vibrato, els legattos massa suaus o algunes notes massa curtes interpretades com a stacatto, que són interpretades per l'algorisme com l'atac de la següent regió.

5.3 – Conclusions de les proves realitzades

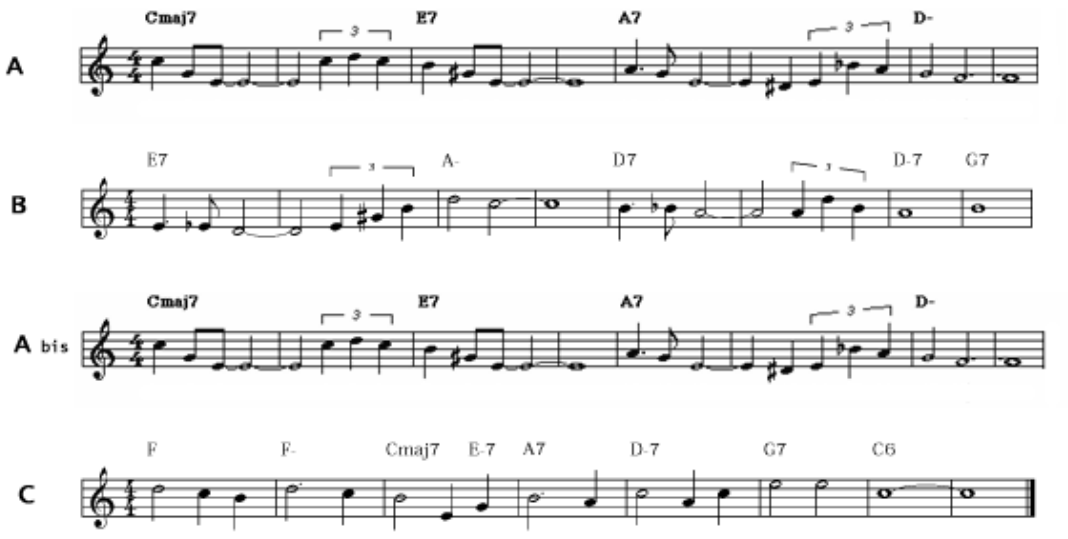
Com ja s'ha dit al capítol 3, l'objectiu més important del projecte és aconseguir uns resultats bons sobretot als casos inexpressius, ja que en els casos expressius hi ha unes limitacions que requereixen d'una anàlisi més complexa on s'extreguin característiques expressives del so, com podria ser el *vibrato* o el *glissando*, per tal de fer mes fàcil la segmentació en regions. Un tema en el que s'està treballant actualment, és precisament el d'extraure el vibrato del so, per tal de tenir una representació del *pitch* sense el *vibrato*, que permet fer la segmentació en regions d'una forma més acurada i tenir la informació del *vibrato* guardada com un atribut d'alt nivell. Aquesta aportació millorarà considerablement els resultats de la segmentació de les melodies pels casos expressius.

Veiem ara a mode de resum els resultats que hem obtingut. En total s'han analitzat una quantitat de 598 notes en les interpretacions no expressives i s'han segmentat correctament 595, això suposa un percentatge d'encert de 99.5%. La quantitat de notes en les interpretacions expressives ha estat de 670 i s'han segmentat correctament 609, el que suposa un 90.9% d'encert.

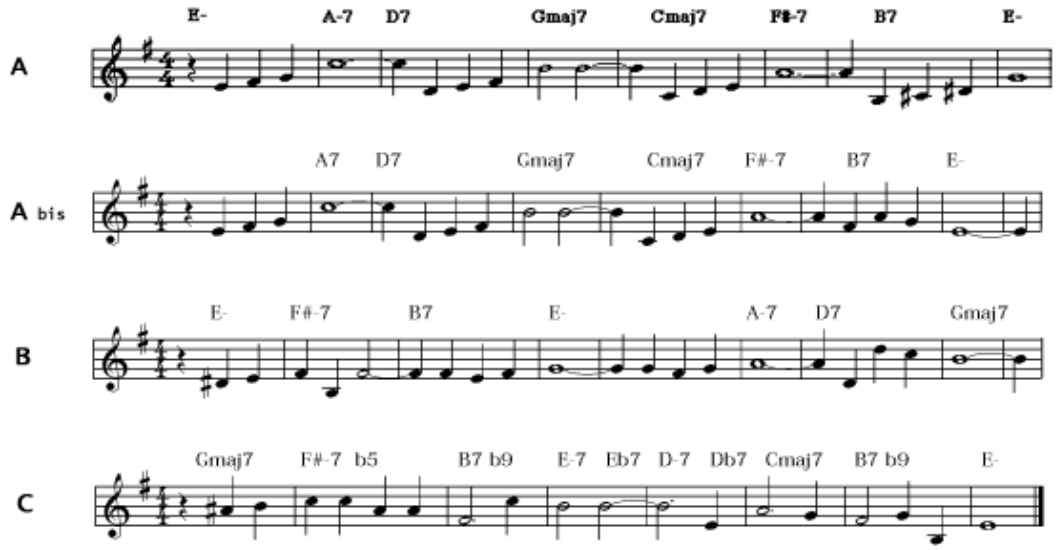
L'elevat percentatge d'encert pels casos no expressius, garanteix la fiabilitat de la integració d'aquest projecte amb els altres dos projectes *SaxEx* i *SMS*.

5.4- Partitures

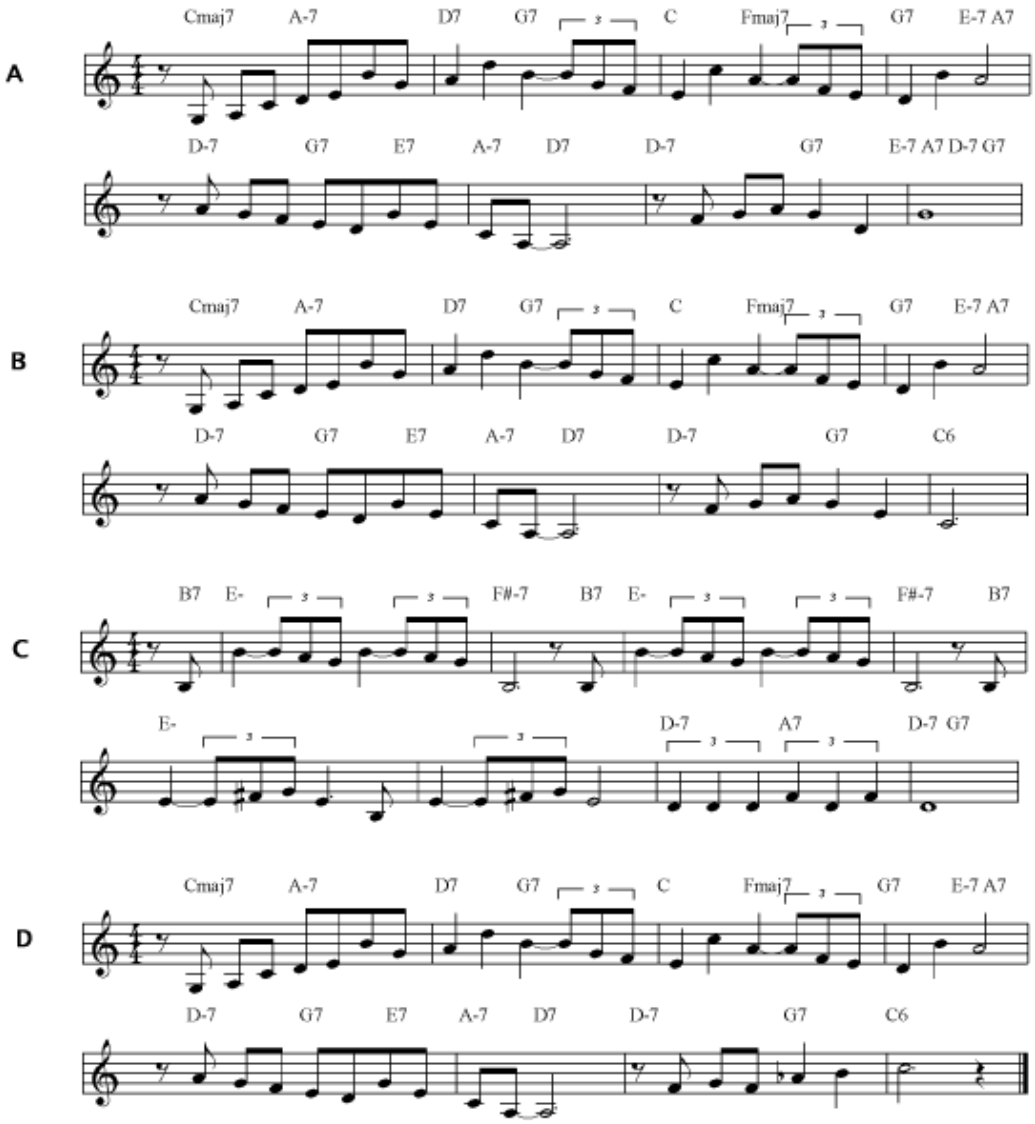
All Of Me



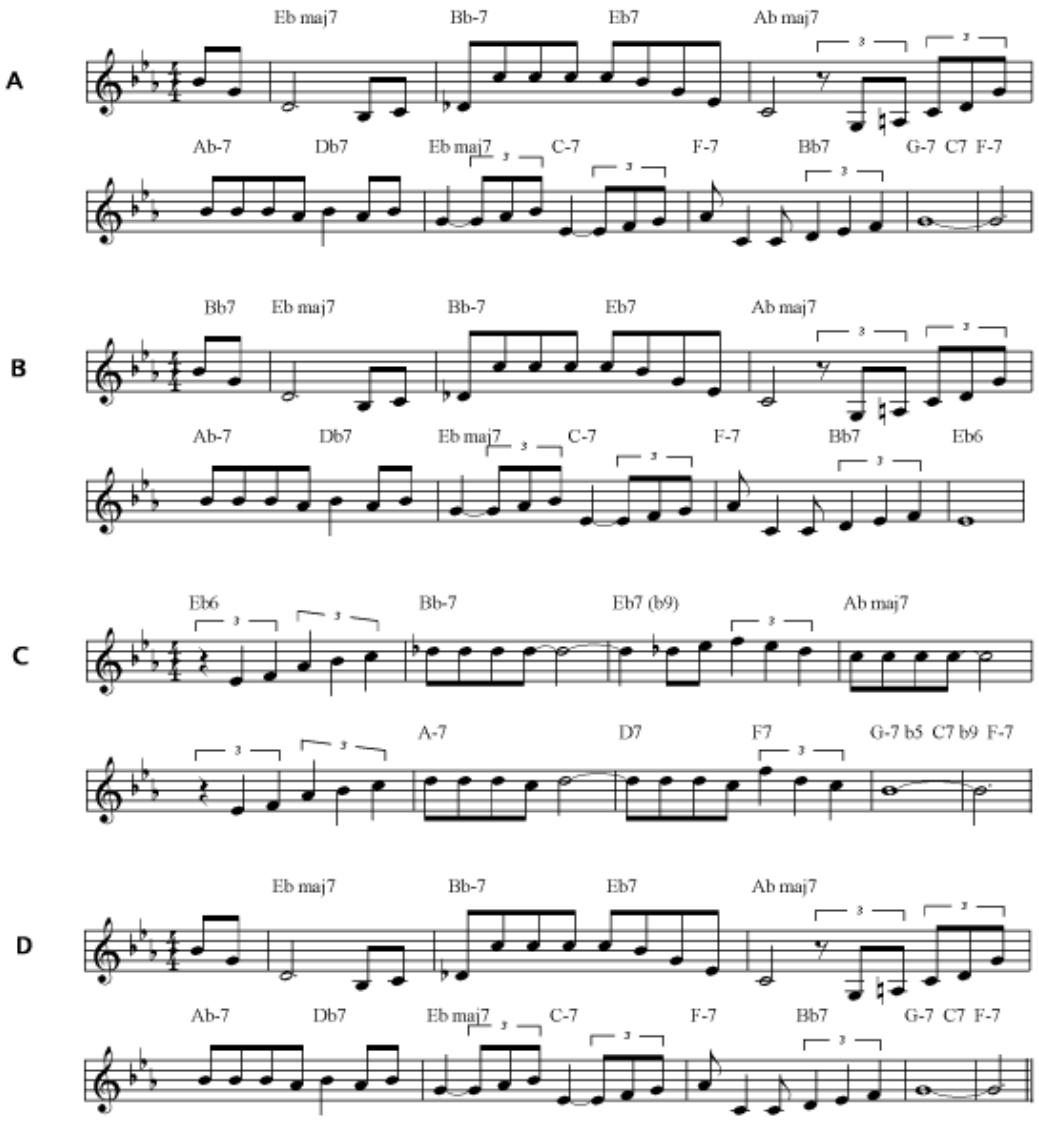
Autumn Leaves



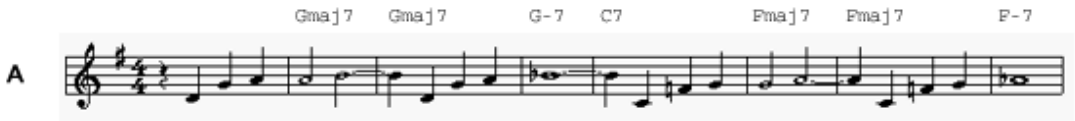
My One and Only Love



Misty



How High The Moon



Capítol 6: Conclusions

En la realització d'aquest projecte, s'ha fet servir el coneixement musical i informàtic, per tal d'assolir l'objectiu proposat a l'inici. Es tractava d'aportar una sèrie de millores i funcionalitats noves a dos projectes existents (*SaxEx* i *SMS*), i fer un enllaç de connexió entre ells, per poder fer ús d'ells veient-los de forma transparent com un de sol. Les aportacions que s'han fet es podrien resumir com:

- S'han afegit una sèrie de paràmetres nous d'anàlisi a *SMS* i s'han ajustat altres per tal d'aconseguir uns resultats millors en l'anàlisi d'instruments monofònics de vent.

- S'han fet millores en la detecció de la freqüència fonamental (*pitch*) de les notes de les melodies analitzades.

- S'ha fet una segmentació de les melodies en regions (*attack*, *steady state*, *release*, *silenci*), per tal de tenir informació de quines notes musicals hi ha al fitxer analitzat, així com la duració, la posició en el temps o la seva articulació.

- S’ha dissenyat un software anomenat *ScoreMan* amb la funció de fer de link entre *SaxEx* i *ScoreMan*. Ens permet fer la síntesi de nous sons expressius a partir d’uns canvis determinats proposats pel sistema *SaxEx*, així com visualitzar diferents tipus de fitxers i fer tot el procés amb un complet sistema de menús o de forma remota via TCP/IP amb una implementació de sockets.

Com s'ha vist al capítol de proves, s'han obtingut uns resultats molt bons en la fase de segmentació en regions, que permeten el bon procés posterior d'expressivitzacció de la melodia que analitza el sistema. L'objectiu futur del projecte és integrar tot aquest procés en un estudi de post-producció d’àudio, de manera, que un músic interpreti una melodia i després, ell mateix com a usuari, amb uns controls pugui escollir el tipus d'expressivitat que vol afegir al fragment gravat, escoltar diferents resultats i decidir amb quin es vol quedar o si vol mantenir l'original. Aquest procés correspondria a analitzar la melodia amb *SMS*, extraure la informació sobre les regions, introduir aquesta informació al procés de raonament basat en casos de *SaxEx*, definint els controls expressius amb que es vulgui dotar al nou so i la sortida de *SaxEx*, passar-la a *ScoreMan*, per tal de que generi el nou so expressiu, llavors l’usuari, un cop escolta el nou so decideix si el vol conservar o vol generar un altre.

Fins al moment de començar el projecte, aquest procés només es podia fer de forma automàtica en el cas de treballar amb fitxers midi, ja que sinó no es tenia informació sobre les regions del fitxer original, però amb aquest projecte s'ha resolt aquest problema, i també s'ha afegit la possibilitat de veure d'una forma clara i entenedora amb una complerta interfície gràfica, els resultats parcials de l'anàlisi, fase de raonament i síntesi, per que usuaris més avançats puguin tenir més control sobre els processos que s'estan fent.

Aquest projecte no es dona per finalitzat en aquest moment, ja que un pas molt important que caldria fer a continuació, seria extraure atributs d'alt nivell a la fase d'anàlisi com podria ser el vibrato, que permetria una representació més complerta de les melodies analitzades, tenint per una banda les notes musicals, i per altra banda els atributs expressius, és a dir és tindria una representació de les característiques del só, que ens permetria fer transformacions del so basades en el contingut, que es una forma semblant a com treballem els humans i que permet més fàcilment l'aplicació de procediments d'intel· ligència artificial per tractar el so, com és el raonament basat en casos que es fa servir al projecte SaxEx.

Referències

[1] Lindley, Mark. "Pythagorean Intonation". *New Grove Dictionary of Music and Musicians*, 15:485-487, ed. Stanley Sadie. Washington, DC, 1980.

[2] Barbour, J. Murray. "The Persistence of the Pythagorean Tuning System". *Scripta Mathematica*, 1:286-304.

[3] X. Serra. “Musical Sound Modeling with Sinusoids Plus Noise”. C. Roads, S. Pope, A. Picialli, G. De Poli, editors. *Musical Signal Processing*. Swets & Zeitlinger Publishers, 1997.

[4] J. L. Arcos, R. López de Mántaras, X. Serra. “SaxEx: A Case-based Reasoning System for Generating Expressive Musical Performances”. *Procedings of the International Computer Music Conference ICMC’97*,1997, *Journal of New Music research*, vol. 27, nº3, 194-210, 1998.

[5] J. L. Arcos, R. López de Mántaras. “An interactive Case-Based Reasoning Approach for Generating Expressive Music”. David W. Aha and Héctor Muñoz-Avila, editors, 1999.

[6] Eugene Narmour. “The Analysis and cognition of basic melodic structures: the implication-realization model”. *University of Chicago Press*, 1990.

[7] Fred Lerdahl and Ray Jackendoff. “An overview of hierarchical structure in music”. In Stephan M. Schwanaver and David A. Levitt, editors, *Machine Models of Music*, pages 289-312. The MIT Press, 1993. Reproduced from Music Perception.

[8] P. Cano. “Fundamental Frequency estimation in the SMS Analysis”, *Procedings of the Digital Audio Effects Workshop (DAFX98)”*, 1998.

[9] R. C. Maher and J. W. Beauchamp. “Fundamental Frequency Estimation of Musical Signals using a two-way Mismatch Procedure”, *Journal of the Acoustical Society of America*. (4):2254-2263, 1994.

[10] Richard C. Leinecker and Tom Archer. “Visual C++ 6 Bible”. IDG Books Worldwide, Inc., Foster City, California, USA, 1998.

[11] Bjarne Stroustrup. "The C++ Programming Language. Third Edition" Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, E.U.A., 1997

[12] Chuck Sher, Bob Bauer. “The New Real Book. vols 1-3”. Sher Music Co., 1998