


Universidad Nacional Autónoma de México

Programa de Maestría y Doctorado en Música

Creación de una aplicación de cómputo generadora de melodías atonales siguiendo la Metodología *Modus Novus* de Lars Edlund

Tesis

que para obtener el título de:

Maestro en Música (Tecnología Musical)

Presenta:

Lic. Alfonso Meave Avila

Tutor:

Dr. Felipe Orduña Bustamante

Ciudad de México, agosto de 2007

Nur so viel scheint zur Einleitung, oder Vorerinnerung, nötig zu sein,
dass es zwei Stämme der menschlichen Erkenntnis gebe,
die vielleicht aus einer gemeinschaftlichen, aber uns unbekannten Wurzel
entspringen,
nämlich Sinnlichkeit und Verstand, durch deren ersteren uns Gegenstände gegeben,
durch den zweiten aber gedacht werden.

Kant, Kritik der reinen Vernunft 1.
Einleitung

„Die Welt ist meine Vorstellung:“ –dies ist eine Wahrheit, welche in Beziehung auf
jedes lebende und erkennende Wesen gilt.

Schopenhauer, Die Welt als Wille und Vorstellung.
Erstes Buch

Alles geht, alles kommt zurück; ewig rollt das Rad des Seins.
Alles stirbt, alles blüht wieder auf, ewig läuft das Jahr des Seins.
Alles bricht, alles wird neu gefügt; ewig baut sich das gleiche Haus des Seins.
Alles scheidet, alles grüßt sich wider; ewig bleibt sich treu der Ring des Seins.
In jedem Nu beginn das Sein; um jedes Hier rollt sich die Kugel Dort.
Die Mitte ist überall. Krumm ist der Pfad der Ewigkeit.

Nietzsche, Also sprach Zarathustra
Dritte Teil: Der Genesende

Mi más sincero agradecimiento
al Dr. Felipe Orduña Bustamante y
al Mtro. Francisco Fernández del Castillo.
Sin su apoyo y entusiasmo este proyecto jamás se hubiera logrado.

„Sokrates, treibe Musik!“
Τεχνη-Arte-Kunst

INDICE Página

Abstract

La presente tesis describe la fundamentación, desarrollo, implementación, prueba y resultados de la aplicación de cómputo denominada *ModusXXI*. *ModusXXI* es un generador melódico no tonal basado en la metodología *Modus Novus* de Lars Edlund y desarrollado en lenguaje de programación Java, bajo un método probabilístico de principios formales de generación melódica. *ModusXXI* es capaz de crear de manera aleatoria un número ilimitado de melodías, que sirvan para la realización de dictados o la generación de material de lectura y entonación, de acuerdo a los parámetros establecidos por el usuario (tales como figuras rítmicas, compás, etc.) y al material interválico propuesto por el *Modus Novus*; posibilitando un estudio exhaustivo de esta metodología de lectura y adiestramiento auditivo no tonal.

1. Introducción

El adiestramiento auditivo es una de las asignaturas principales en la formación del músico profesional. Si bien en la actualidad existe una gran variedad de métodos de adiestramiento auditivo, la mayoría de éstos se concentran en el desarrollo de las capacidades auditivas y de lectura musical dentro del sistema armónico-melódico tonal. Esta particularidad en el desarrollo de metodologías didácticas se presenta tanto en los métodos creados en formato de libros de texto, así como en aquellos desarrollados en forma de aplicaciones de cómputo.

Un intento por abordar el adiestramiento auditivo y la lectura de música "no tonal" es el método *Modus Novus* del compositor sueco Lars Edlund. El *Modus Novus* es un libro de texto cuyo objetivo, según el propio Edlund, es «…atacar los problemas relacionados con la lectura de la música del siglo XX que no se encuentra en tonalidad mayor/menor.»[1] El *Modus Novus* basa su metodología en el estudio clasificado del material interválico. Cada capítulo presenta ciertos intervalos, incluidos en el texto de Edlund, para generar a partir de ellos ejercicios melódicos cortos, melodías y series de acordes, los cuales pueden servir como material de entonación o dictados. Un problema durante el estudio del *Modus Novus* es la insuficiencia del material. Los ejercicios preparatorios no llegan a los cincuenta por capítulo y las melodías (salvo los capítulos que son ejemplos del repertorio musical) no llegan a las diez por capítulo. Esta insuficiencia resulta evidente si se considera que dicho material se usa tanto para la lectura como para los dictados.

Es objetivo del presente trabajo desarrollar una aplicación de cómputo que genere material melódico de acuerdo a la metodología propuesta por el *Modus Novus*, y que sirva para la realización de dictados o la generación de material de lectura y entonación durante el estudio de este libro. La aplicación de cómputo desarrollada en esta tesis, que hemos denominado *ModusXXI*, es capaz de crear aleatoriamente un número ilimitado de melodías, de acuerdo a los parámetros establecidos por el usuario y al material interválico propuesto por el *Modus Novus*; posibilitando un estudio exhaustivo de esta metodología de lectura y adiestramiento auditivo no tonal.

[1] Edlund Lars, *Modus Novus*, AB Nordiska Musikförlaget/ Edition Wilhelm Hansen Stockolm, Suecia 1963, pag.3.

Esta tesis está organizada en los siguientes capítulos:

- Antecedentes: Breve exposición sobre el adiestramiento auditivo, la metodología *Modus Novus*, el desarrollo de aplicaciones de cómputo para el adiestramiento auditivo y los procesos de generación de material musical por computadora.
- Desarrollo: Exposición de los aspectos básicos de la teoría de programación e ingeniería de software, así como la implementación de *ModusXXI*.
- Resultados: Organización, funcionamiento y evaluación de la aplicación.
- Conclusiones: Logros y perspectivas de desarrollo ulterior.
- Bibliografía.
- Apéndices:
    - Tabla: Evaluación de Uso y Calidad de *ModusXXI* versión 1.0.
    - Documentación.
    - Código fuente.
    - Ejemplos musicales.

2. Antecedentes

2.1 El adiestramiento auditivo

El adiestramiento auditivo y la lectura de música forman, junto con la clases de teoría musical y ejecución instrumental, los pilares de la educación musical profesional. El adiestramiento auditivo es la asignatura encargada de desarrollar las capacidades de percepción auditiva consciente en el músico. El adiestramiento auditivo busca formar una "conciencia auditiva" donde los conceptos de la teoría musical no son asimilados como meras abstracciones producto de la pura reflexión intelectual, sino que son resultado de la comprensión auditiva-intelectual de los fenómenos sonoro-musicales percibidos. El pedagogo Roland Mackamul nombra al adiestramiento auditivo «La educación de la representación auditiva interna consciente».[1] Como ejemplo valdría la siguiente analogía: así como un pintor es capaz de al cerrar los ojos visualizar internamente una imagen, por ejemplo, una flor, para después reproducirla gráficamente en un lienzo; de la misma manera el músico debiera ser capaz de imaginar o escuchar internamente una melodía o estructura armónico-polifónica, para posteriormente almacenarla y/o reproducirla en algún medio, como por ejemplo una partitura. Necesariamente, transcribir imágenes sonoras internas requiere de un cierto dominio intelectual, o mejor aún, de la asimilación de la teoría musical; pero dicha asimilación debe estar dada de manera paralela al reconocimiento auditivo del material musical, pues de lo contrario, la teoría musical y la percepción auditiva permanecen como dos aspectos de la formación musical aislados e incapaces de establecer contacto entre sí. Al respecto Mackamul señala:

> El adiestramiento auditivo pretende formar conciencia auditiva, para oír conscientemente sonidos y sonidos musicalmente relacionados entre sí. Pretende desarrollar en el alumno la representación de la escritura, la audición y ejecución musicales, integrándolas en una imagen global, la "audición interna". Pretende capacitarlo para que conscientemente pueda captar, retener, y reproducir los sucesos musicales; más concretamente: el alumno debe aprender a reconocer y nombrar los sonidos y sus relaciones contextuales por él escuchados, o sea, a definirlos técnicamente. Debe aprender a traducir ejemplos musicales a signos musicales con la ayuda de dicha información. De la misma manera, esa definición debe hacerlo capaz de reproducir en su instrumento lo que haya escuchado. Además debe poder traducir en sonido lo que perciba por la vista, es decir, debe poder cantar un texto musical y debe poder imaginar cómo suena una partitura compleja.[2]

[1] Mackamul Roland, *Sensibilización al fenómeno sonoro*, Cátedra Extraordinaria Manuel M. Ponce, SECEP/UNAM, México 1982, p.18.

[2] Ibidem, p.18.

Tomando en cuenta lo señalado por Mackamul, podríamos considerar los siguientes aspectos como objetivos prácticos esenciales del adiestramiento auditivo:

1. Reconocer y nombrar los sonidos y sus relaciones contextuales.
2. Transcribir ejemplos musicales sonoros a signos musicales.
3. Reproducir dichos ejemplos musicales con algún instrumento.
4. Cantar un texto musical (partitura).
5. Imaginar partituras complejas.

Como asignatura obligatoria, el adiestramiento auditivo fue introducido a partir de la posguerra (1945) en los conservatorios y escuelas superiores de música de los países germano-hablantes y llevado a los Estados Unidos por músicos provenientes de Alemania como Paul Hindemith.[3]

Durante el siglo XX, y lo que va del XXI, han surgido alrededor del mundo una gran cantidad de métodos de entrenamiento auditivo. Sin embargo, la mayoría de estos métodos han estado enfocados a la música denominada "tonal". La música tonal es aquella que se desarrolla básicamente en Europa entre los siglos XVII, XVIII y XIX, y que estilísticamente pertenece al barroco, clasicismo y romanticismo. El sistema tonal es un sistema armónico-melódico donde los grados de las escalas mayor y menor (natural, armónica y melódica) van a establecer diferentes relaciones respecto al sonido denominado tónica. Más importante aún que las funciones melódicas van a ser las funciones armónicas; de hecho, es el acorde de dominante o V_7 ,y su resolución, la pieza fundamental del sistema tonal. Debido al dominio que ha tenido en el repertorio de las orquestas y demás ensambles de música académica la música escrita entre los siglos XVII y XIX, la pedagogía musical ha limitado su campo de acción a la música tonal. Si bien ya en el siglo XIX, y sobretodo durante el XX, una explosión de nuevos sistemas de composición melódico-armónicos se gestó, pocos han sido los intentos de un acercamiento didáctico a los géneros distintos al sistema tonal.

[3] Mackamul Roland, *Sensibilización al fenómeno sonoro*, Cátedra Extraordinaria Manuel M. Ponce, SECEP/UNAM, México 1982, p.17.

Uno de estos pocos intentos de acercamiento a la música no tonal es el método de entrenamiento auditivo y lectura de música *Modus Novus* del compositor sueco Lars Edlund.

2.1.1 El *Modus Novus*: Presentación

El libro *Modus Novus* es el primero de dos textos didácticos enfocados al adiestramiento auditivo y la lectura musical escritos por el compositor sueco Lars Edlund (1922), durante su estancia como profesor de entrenamiento auditivo en la Real Academia de Música de Estocolmo. Publicado por primera vez en 1963 el *Modus Novus*, señala Edlund:

> …es un intento de atacar los problemas relacionados con la lectura de la música del siglo XX que no se encuentra en tonalidad mayor/menor...Este libro está dirigido primeramente a los estudiantes de música y trata en primera instancia con la lectura de la música del siglo XX.[4]

El *Modus Novus* es el primer método de entrenamiento auditivo que sistematiza de cierta forma la enseñanza de la música no tonal. Al respecto apunta Edlund:

> El entrenamiento auditivo tradicional ha estado basado en el sistema de tonalidad mayor/menor, y su objetivo ha sido el desarrollar un dominio de la música tonal mayor/menor. Los métodos aplicados en el entrenamiento auditivo han estado dictados por el material musical, y durante esta época se han basado en las leyes de la cadencia tonal.[5] [Sin embargo, continua Edlund] con el 'colapso' del sistema tonal mayor/menor aquellos que están relacionados con el adiestramiento auditivo han sido confrontados con una situación completamente nueva. El entrenamiento auditivo convencional no satisface los requerimientos de la música del siglo XX.[6]

Para comprender mejor esta situación analicemos la estructura y concepción de la otra obra didáctica de Edlund: el *Modus Vetus.*

---

[4] Edlund Lars, *Modus Novus,* pag.3, AB Nordiska Musikförlaget/ Edition Wilhelm Hansen Stockolm, Suecia 1963.

[5] Ibidem p.13 (Traducción del autor).

[6] Ibidem p.13.

2.1.2 *Modus Vetus*: El "modo viejo", la tonalidad vista desde el siglo XX

El *Modus Vetus* es el "hermano menor" del *Modus Novus*. Está obra fue publicada en 1967, posteriormente al *Modus Novus* y es, en contraparte a éste último, una obra dedicada al entrenamiento auditivo y la lectura dentro de un sistema tonal. El *Modus Vetus*, 'Modo Viejo'[7] como lo llama el mismo Edlund, es un estudio conciso de la música que va del barroco y llega hasta el romanticismo Wagneriano. Está dividido en cuatro secciones: ejercicios de lectura melódica, ejercicios rítmicos, ejercicios de bajo cifrado y ejercicios de armonía al teclado.[8] Cada una de estas secciones está construida en función del sistema de tonalidad mayor-menor, y a diferencia de muchos métodos, toma ejemplos de la literatura musical con el fin de que el alumno se relacione directamente con los diferentes estilos que el sistema tonal posee. Para Edlund, el *Modus Vetus* es «un libro publicado dos siglos demasiado tarde.»[9] No obstante, considera « resulta aún necesario el estudio metodológico de este tema. Mientras Bach, Mozart y Beethoven sean tocados y signifiquen mucho para nosotros, estamos obligados a estudiar su 'vocabulario.»[10]

Si bien para el adiestramiento auditivo resulta imprescindible el conocimiento y dominio metodológico del sistema tonal, las herramientas que han sido creadas para dicho sistema no necesariamente van a funcionar en los contextos no tonales. Por ello la necesidad de crear métodos de adiestramiento auditivo que respondan a las necesidades de la 'Nueva Música'.

2.1.3 *Modus Novus*: Perspectivas nuevas en el siglo XX

Dado que los métodos de entrenamiento auditivo tradicionales no ofrecen soluciones a los problemas que presenta la música no tonal, ¿cómo es que estos nuevos métodos deben estar organizados? La respuesta no es fácil, pues en la música contemporánea existen un sin fin de formas de organización del material sonoro las cuales no siempre pueden agruparse dentro de un sistema o metodología de estudio. En este sentido se pregunta Edlund: «¿posee la música compuesta en el siglo XX alguna estructura

---

7 Edlund Lars, *Modus Vetus*, Prefacio, AB Nordiska Musikförlaget/ Edition Wilhelm Hansen Stockolm, Suecia 1967.

8 Edlund Lars, *Modus Vetus*, Prefacio (Traducción del autor).

9 Edlund Lars, *Modus Vetus*, Prefacio.

10 Edlund Lars, *Modus Vetus*, Prefacio.

lógica que sirva de base para el desarrollo de un método de entrenamiento auditivo?»[11] La respuesta es contundente: No. El campo de estudio propuesto por Edlund en el *Modus Novus* es limitado y solamente recoge algunas de los estilos desarrollados durante la primera mitad del siglo XX. Al respecto apunta:

> El material de estudio presentado en este libro ha sido diseñado en base a un número finito de sonoridades y figuras melódicas que, en opinión del autor, han jugado un rol importante en la ruptura de la música del siglo XX con la tonalidad...los problemas de lectura y audición conectados con la llamada 'música radical' no son tratados aquí.[12]

Resulta necesario aclarar los alcances que ofrece el *Modus Novus* como método de entrenamiento auditivo.

2.1.4 *Modus Novus*: Alcances

Como se mencionó, el *Modus Novus* es un método limitado, que sólo ofrece acercamiento a un cierto sector de la música del siglo XX. Este sector abarca el atonalismo serial de la segunda escuela vienesa, liderado por Arnold Schönberg, Alban Berg y Anton Webern y la "no tonalidad" menos radical de compositores como Stravinsky, Bartok y Hindemith entre otros. El *Modus Novus* no aborda la música microtonal, ni toda la gama de estilos derivados de la música electrónica. Este método de adiestramiento auditivo se limita a la música creada dentro de un sistema de doce semitonos cromáticos de temperamento igual que busca romper todas las relaciones tonales posibles en la música.

Dado su corto campo de acción, ¿en qué radica la importancia de este método de adiestramiento auditivo, si solamente atiende a un pequeño sector musical; el cual además pareciera que ha detenido su desarrollo en el ya iniciado siglo XXI? La importancia del *Modus Novus* radica, a nuestro juicio, en que es el primer método que asume a los intervalos musicales como entidades generadoras de material musical. En el *Modus Novus* las funciones tonales son sustituidas por las 'funciones interválicas'; pues son ahora los

[11] Edlund Lars, *Modus Novus*, pag.13.

[12] Edlund Lars, *Modus Novus*, pag.13.

intervalos y sus ‘relaciones’ lo que da cohesión al discurso musical. Al respecto apunta Edlund:

> Una de mis tesis principales es que la capacidad de cantar correctamente intervalos aislados no siempre es una garantía de precisión a la hora de cantar melodías atonales...[13] la tarea más importante del entrenamiento auditivo es ahora practicar las ‘combinaciones de intervalos’ que rompen las barreras de la interpretación tonal que cada intervalo pudiese tener.[14]

A este estudio de las relaciones no tonales entre los intervalos Edlund le va a llamar «Estudio auditivo de los patrones musicales».[15]

Quizá la aportación más importante que hace el *Modus Novus* al adiestramiento auditivo es el rompimiento real con la tonalidad, yendo más allá del estudio de los intervalos como unidades separadas y, sobretodo, estableciendo un nuevo sistema de relaciones o funciones en la música; donde el eje principal ya no es la ‘tónica’ sino la interválica que se desarrolla tanto en el contexto melódico como en el armónico. Así, para el *Modus Novus* son las relaciones interválicas, no los intervalos, lo que va a dar cohesión al discurso musical.

2.1.5 Estructura del Modus Novus

Edlund nos ofrece una rápida y simple descripción de la organización del *Modus Novus*: «las figuras melódicas han sido agrupadas, en relación a los intervalos que contienen, en capítulos cuyo grado de dificultad va en aumento.»[16] El *Modus Novus* está organizado en doce capítulos de la siguiente forma:

---

[13] Edlund Lars, *Modus Novus*, pag. 13.

[14] Edlund Lars, *Modus Novus*, pag.13.

[15] Edlund Lars, *Modus Novus*, pag.14.

[16] Edlund Lars, *Modus Novus*, pag.13.

Capítulo	Material Interválico
I	2a menor, 2ª mayor y 4ª justa
II	5a justa + el material precedente
III	3a menor, 3ª mayor + el material precedente
IV	Ejemplos melódicos de la literatura musical aplicando los materiales de los capítulos I, II y III
V	Tritono (4a aumentada o 5a disminuida) + el material precedente
VI	6a menor + el material precedente
VII	6a mayor + el material precedente
VIII	Ejemplos melódicos de la literatura musical aplicando los materiales de los capítulos V al VII
IX	7a menor + el material precedente
X	7a mayor + el material precedente
XI	Ejemplos melódicos de la literatura musical aplicando los materiales de los capítulos IX y X
XII	Intervalos compuestos “Weitmelodik”

Cada capítulo posee la siguiente estructura:

Apartados de cada capítulo
1. Presentación del material del capítulo
2. Ejercicios preparatorios
3. Melodías
4. Series de acordes
5. Ejemplos melódicos de la literatura musical

Los *ejercicios preparatorios* consisten en un promedio de 30 frases pequeñas construidas con los intervalos en cuestión. Las *melodías* son ejemplos más largos, y al igual que los *ejercicios preparatorios* están construidas en base a los intervalos de estudio. Las *series de acordes* presentan material armónico basado en los intervalos de estudio. Los *ejemplos melódicos de la literatura musical* tienen como objetivo relacionar al alumno con

casos reales de aplicación del material interválico estudiado. Queda aquí patente la intención de Edlund de, al igual que en el *Modus Vetus*, establecer vínculos entre su metodología y la práctica musical real.

Si bien la didáctica del *Modus Novus* abarca los aspectos melódico y armónico de la música, su punto medular es el aspecto melódico. En general establece la lectura entonada de los ejemplos melódicos y el dictado como metodologías principales. El aspecto armónico no es tratado tan fuertemente como el melódico. Las series de acordes son muy limitadas y ofrecen una visión bastante resumida de las relaciones armónicas no tonales. El ritmo es un aspecto que no es objeto de estudio del *Modus Novus*. Esto no quiere decir que no aparezca este elemento dentro de los ejemplos, sino que su estudio es dejado de lado y al estudiante se le recomienda remitirse al método *Laerebog i Rytmeläsning* de JØrgen Jersild.[17] El manejo de la métrica se basa, en la mayoría de los caso, en los compases convencionales y la escritura se basa en el sistema de notación musical tradicional.

2.1.6 Sobre el autor: Lars Edlund

Existe poca información publicada sobre Lars Edlund y sus libros poseen muy poca información personal. Asimismo, existe escasa, o nula, bibliografía referente a él y en Internet solamente algunos catálogos de compositores ofrecen información referente a Edlund. Según el catálogo de música *operone.de*[18] Edlund nació el 6 de noviembre de 1922 en Suecia. Estudió en la Schola Cantorum de Basilea y se desempeñó como músico en la iglesia luterana, hasta convertirse posteriormente al catolicismo. Trabajó como profesor del *Royal College of Music* en Estocolmo y fue en este tiempo durante el cual escribió sus libros *Modus Novus* y *Modus Vetus*. En 1971 dejó Estocolmo y se trasladó a la isla báltica de Gotland para dedicarse de tiempo completo a la composición. En los 80´s regresó al continente, a la ciudad sueca de Uppsala, continuando su labor como compositor.[19] Hasta aquí la información obtenida respecto a Lars Edlund.

---

[17] JØrgen Jersild, *Laerebog i Rytmeläsning*, Copenhagen 1962.

[18] http://www.operone.de/

[19] http://home.swipnet.se/sonoloco7/PhonoSuecia/edlund.html

2.2 *Adiestramiento auditivo para la música del siglo XX* (*Ear Training for Twentieth-Century Music*): un método alternativo al *Modus Novus*

Adiestramiento auditivo para la música del siglo XX (*Ear Training for Twentieth-Century Music*) de Michael Friedmann es un método alterno al *Modus Novus*. Publicado en 1990, este libro hace un análisis auditivo-estructural de la música no tonal de la primera mitad del siglo XX, basado en la metodología de análisis musical propuesta por Allen Forte en su libro *The Structure of Atonal Music*[20]. El método de Friedmann se compone de seis capítulos:

1. *Calisténicos.*
2. *Diadas* (Dyads): Movimiento melódico y estructura armónica.
3. *Procesos:* Altura (pitch), clases de altura (pitch class) y relaciones de contornos.
4. *Triadas:* conjuntos de tres elementos.
5. *Tetracordes:* conjuntos de cuatro elementos.
6. *Conjuntos de más de cuatro elementos.*

Los "calisténicos musicales" señala Friedmann, «son ejercicios para facilitar la memorización que carece de los beneficios de cualquier estructura lingüística.»[21] Así, el capítulo primero es una serie de ejercicios de "gimnasia musical", los cuales «presumen la habilidad [por parte del alumno] de repetir melodías vocalmente y su capacidad para escribirlas.»[22] Dichos ejercicios consisten en toma de dictados y repetición por imitación, ya sea vocal o instrumental, de una serie de ejemplos melódicos.

El capítulo 2 "Diadas" inicia con una serie de definiciones de conceptos sobre los que Friedmann basa el análisis auditivo de su método. Una "Diada" «es una unidad armónica constituida por dos tonos»[23]. Es decir, una diada es un intervalo. A la par, define Friedmann altura y otra serie de conceptos que utiliza para su análisis estructural-auditivo. Un concepto básico en el método de Friedmann es el de "clase de altura" o *pitch class*. Este

[20] Forte Allen, *The Structure of Atonal Music*, Yale University Press, USA 1973
[21] Friedmann Michael L., *Ear Training for the Twentieth-Century Music*, Yale University Press, USA 1990, pag. xxi.

[22] Ibidem pag.3 (Traducción del autor).

[23] Ibidem pag. 9.

concepto se refiere a la nomenclatura de los sonidos de una escala cromática. En este caso, utiliza números para nombrar los doce semitonos cromáticos en lugar de utilizar la nomenclatura latina (do, re, mi, etc.) o la nomenclatura anglo-sajona (a, b, c, etc.). Según el propio Friedmann "conjunto de clases" (de altura) o *set class* «es la estructura armónica fundamental de la música post-tonal.»[24] El "conjunto de clases" «es una categoría para sonoridades compuestas de dos a diez "clases de altura."»[25] Cabe señalar que el "conjunto de clases" es un concepto que abarca tanto las relaciones armónicas como las relaciones melódicas entre los sonidos. Otro concepto importante es el de "intervalo de clase de altura sin orden" (*unordered pitch class interval*), el cual reduce los intervalos a seis clases:

1. segundas menores-séptimas mayores
2. segundas mayores-séptimas menores
3. terceras mayores-sextas menores
4. terceras menores-sextas mayores
5. cuartas justas-quintas justas
6. tritono

Respecto a esta clasificación señala Friedmann:

> Los seis "intervalos de clase de altura sin orden" son el punto de referencia más general y abstracto de las relaciones entre dos alturas. En lugar de pensar un "intervalo de clase de altura" como una distancia entre dos notas capaz de ser medidas, podría ser de más ayuda el pensarlo como un tipo de sonoridad, análogo a un "color" o timbre.[26]

Esta clasificación del material interválico por parte de Friedmann es el punto de encuentro más cercano con el *Modus Novus*. Parte del análisis de Friedmann se refiere a la descripción del movimiento armónico-melódico basado en estos "intervalos de clase de altura". No obstante, a diferencia del *Modus Novus,* para Friedmann el material interválico es sólo un elemento más en su metodología de análisis estructural-auditivo.

---

[24] Ibidem pag.39.

[25] Ibidem pag.14.

[26] Ibidem pag.15.

El capítulo tres presenta los conceptos más trascendentes del método de Friedmann, ya que define los procesos de generación de material musical más comunes en la música atonal del siglo XX. Dichos procesos son:

1. Retrogresión
2. Transposición
3. Inversión de altura
4. Inversión de clases de altura
5. Invariabilidad (*Invariance*)
6. Series de contornos adyacentes
7. Clases de contorno

La *retrogresión* se refiere básicamente a un palíndromo, la *transposición* es un cambio de altura de ciertas relaciones interválicas, la *inversión* es el cambio simétrico de dirección melódica de un intervalo sobre una nota "axial"; por ejemplo, la inversión de una segunda menor ascendente, sobre la nota axial, sería una segunda menor descendente. La *inversión de clases de altura* es igual que la inversión de alturas, sólo que aplicado a grupos de notas más grandes. La *invariabilidad* se refiere a los elementos comunes entre las unidades musicales relacionadas a un proceso como la transposición o la inversión. Por ejemplo, la nota "axial" o nota pivote de dos procesos. Las *series de contornos adyacentes* se refiere «a una serie de signos de '+ ó –' correspondientes a la dirección melódica ascendente o descendente. La *clase de contorno* ordena las alturas de una unidad musical donde el sonido más grave va a ser denominado '0' [cero].»[27]

Los capítulos cuatro, cinco y seis (triadas, tetracordes y conjuntos de más de cuatro elementos) son básicamente la aplicación de los conceptos del capítulo tres a grupos de tres o más sonidos. Dicha aplicación resulta un análisis estructural-auditivo tanto de las relaciones melódicas como armónicas de la música no tonal.

[27] Ibidem páginas 33 y 34.

La diferencia metodológica entre el *Modus Novus* y el libro de Friedmann es considerable, pues mientras que el *Modus Novus* se basa en el estudio de las relaciones interválicas no tonales en la música del siglo XX, el método de Friedmann pondera el aspecto estructural y los procesos de generación de la música no tonal. Cabe señalar que estos procesos de generación de material musical (retrogresión, transportación, inversión, etc.) podrían ser implementados en un futuro a la aplicación de cómputo *ModusXXI*, lo cual daría pie a la fusión de ambas metodologías ofreciendo mejoras al adiestramiento auditivo de la música no tonal del siglo XX. No obstante dicha implementación sale de los objetivos y alcances del presente trabajo.

2.3 Las aplicaciones de cómputo para el adiestramiento auditivo

A partir de la década de los ochentas se generó toda una revolución informática en el campo de la música. El desarrollo del protocolo MIDI y del audio digital se convirtieron en los ejes de desarrollo de tecnologías musicales que han impactado la creación, ejecución, grabación y didáctica musicales. Hoy día, principios del siglo XXI, la informática musical se ha apoderado por completo del quehacer musical comercial. En el caso de la música 'culta', 'clásica', de 'conservatorio' o como se le desee nombrar a aquella música que proviene de las universidades y escuelas superiores de música, el ingreso de la tecnología ha sido más lento pero definitivo. Específicamente es en el ámbito de la composición donde el desarrollo informático ha logrado mayores espacios. No obstante, poco a poco la pedagogía musical se ha interesado por los beneficios que ofrece el uso de las tecnologías informáticas.

En el caso del adiestramiento auditivo, encontramos programas didácticos de cómputo enfocados a esta área desde finales de los ochentas. Como ejemplo se encuentran *Ear Trainer* (1989, desarrollado por Lawrence Gallagher) para plataforma Macintosh y *GUIDO 2.1* (1989) para sistema operativo DOS. Desde entonces se ha desarrollado un número considerable de programas de cómputo para el adiestramiento auditivo. Actualmente es posible encontrar una oferta relativamente amplia de programas didácticos musicales; no obstante, pocos son los que ofrecen metodologías didácticas de calidad. Prácticamente todos los programas de cómputo para el adiestramiento auditivo cuentan con

una didáctica tonal basada en ciertos principios de la armonía tonal 'clásica' y la armonía de jazz. Esto tiene una obvia razón, y es la intensión preponderantemente comercial de las empresas que crean estos programas sobre los objetivos académico-didácticos, base del conocimiento universitario. Tanto las empresas de programación como las universidades, han tenido un interés casi nulo por desarrollar aplicaciones de cómputo didácticas que faciliten el acercamiento a las géneros musicales que no son los 'tradicionales' o comerciales.

2.3.1 Programas de computo para el adiestramiento auditivo existentes

Sin duda actualmente existe una gran cantidad de programas de cómputo para el adiestramiento auditivo. En la tesis titulada *Computer-assisted instruction in ear-training and its integration into undergraduate music programs during the 1998-99 academic year*[28] *(Instrucción asistida por computadora en el entrenamiento auditivo y su integración en los programas de licenciatura durante el año académico 1998-99)* presentada en 1999 por Douglas Raimond Spangler para obtener el grado de Maestro en Música en la Universidad del Estado de Michigan (EE.UU.), se hace un listado de más de 60 aplicaciones de cómputo para el entrenamiento auditivo y se hace una revisión de 30 de éstos, así como los resultados obtenidos en la integración de algunos de ellos en los programas de estudio de algunas licenciatura en Música en las Universidades de los Estados Unidos de América. A continuación es presentado el listado de aplicaciones, elaborado por Spangler:

Programas Recientes (1998)	**Sistema Operativo**
Anvil Studio (1999 MIDI sequencer/eartraining)	Windows
Auralia (1998)	Windows
Aural Skills Trainer (1998)	MAC/WIN
Chord ID (1998)	Windows
Dolphin Don's Music School (1998)	Windows

[28] http://www.msu.edu/user/spangle9/thesis.html

EarMaster & EarMaster School (1998)	Windows
Earobics (1998)	Windows
EarPower (1998)	Windows
Eartraining (1998)	Macintosh
ETDrill (1999)	Windows
Fanfare (1998)	Windows
Harmonic Hearing (1998)	Macintosh
Harmonic Progressions (1999)	MAC/WIN
HearMaster (1998)	MAC/WIN
Inner Hearing (1999)	MAC/WIN
Listen (1998)	Macintosh
MacGAMUT (1998)	Macintosh
MiBAC Music Lessons (1998)	MAC/WIN
Musicianship Basics (1999)	MAC/WIN
Music Lab--Harmony (1999)	Windows
Music Lab--Melody (1999)	MAC/WIN
Personal Ear Trainer (PET) (1998)	Windows
Pitch ID (1998)	Windows
Practica Musica (1999)	Macintosh
Teoría (1998)	Windows
The Music Box (1999)	Windows
Software Libre y basado en Internet	**O/S**
AudioChallenger (Anthony Holland)	NeXT
BigEars (Java Program)	web-based
Chordtrainer (still checking: random 3 note chords)	Windows
EARTRAINER 1.0@web (website)	web-based
Ear Trainer (1989 by Lawrence Gallagher)	Macintosh
Ear Training Website (online exercises/quizzes)	web-based
Ear Training Website (by Jack Haynes)	web-based
ET drills (1996 Quicktime drills)	MAC/WIN

E-Train (a melody game by Victor Grauer)	DOS
Tour-Part Dictation 5.1 (1996 Hypercard)	Macintosh
Halves/Not Halves (1998)	Windows
Ike's Ear Tuner 1.1	Windows
Just Intonation Ear Trainer (1996 Hypercard)	Macintosh
Melodic Ear (May 1999 NEW FREEWARE)	Windows
WebEartraining	web-based
Programas ligeramente viejos	**O/S**
Aural (1994 by Mark Grimshaw)	Atari (ST)
CAETS (1996)	Windows
C.A.T. (ca. 1995 Hypercard)	Macintosh
Ear Challenger (ECS media)	Windows
EARTEST (1995)	Windows
Ear Training: A Technique for Listening (ca.1995)	Macintosh
Ear Trainer (1997)	DOS
GUIDO 2.1 (1989)	DOS
Listen!-A music skills... (ECS media)	DOS
Patterns in Pitch (ECS media)	MAC/WIN
Play it by Ear (ca.1995)	Windows
Rhythmaticity Advanced (ca.1995)	DOS
Rhythmaticity Basic (ca.1995)	DOS
Super Ear Challenger (ECS media)	MAC/DOS
Take Note (1997)	Windows
Toon Up (ECS media)	Windows
Tune-it II (ECS media)	MAC/DOS
WinOye (Now teoria--see above)	Windows
Programas por salir al mercado (1999)	**O/S**
CALMA	MAC/WIN
Computerkolleg Musik	Windows
Hearing Tonal Harmony	Macintosh

NoteWell (Nov.1999)	Macintosh
Thought Sauce Ear Training	DOS
Programas relacionados	**O/S**
A Musical Tutorial (1999)	Windows
Common-Practice (written music theory)	Macintosh
Explorations (written music theory)	Macintosh
Programas para niños	**MAC/WIN**
Music Mastery (1998 writing/composing)	Windows
Musicware Piano	Windows
Singing Tutor	Windows
The Music Kit (written music theory)	Macintosh

Spangler revisa 27 programas de adiestramiento auditivo de los cuales ninguno se ocupa de la música no tonal.

Hoy en día, principios del 2007, existe un número similar de programas de adiestramiento auditivo disponibles o anunciados en Internet. Algunos de los mencionados anteriormente han continuado su desarrollo, otros han dejado de actualizarse y algunos nuevos han surgido (i.e. Perfect Ear[29]); no obstante el perfil de los programas no ha cambiado y todos ofrecen únicamente soluciones a los problemas de la música tonal y el jazz.

[29] http://jmusic.ci.qut.edu.au/applications.html (Agosto 2008)

2.3.2 Análisis funcional de dos programas de entrenamiento auditivo: Auralia 3.0 y EarMaster 5.0

A continuación se presenta una breve descripción, en su versión más reciente (2006), de dos de las aplicaciones de cómputo más populares para el adiestramiento auditivo: Auralia y EarMaster.

Auralia (para sistemas operativos Windows y Machintosh)[30]

Auralia es un programa de adiestramiento auditivo creado por la compañía *Rising Software* en Australia. Este programa en su más reciente versión 3.0 para Windows (la más reciente versión para Machintosh es 2.1) está organizado en 5 secciones: Intervalos y escalas, ritmo, altura y melodías, acordes, armonía y formas musicales.

Secciones y temas que abarca el programa Auralia 3.0 para Windows

Intervalos y escalas	Ritmo	Altura y melodías	Acordes	Armonía y formas
1. Comparación de intervalos 2. Imitación de intervalos 3. Reconocimiento de intervalos 4. Escalas de jazz 5. Entonación de escalas de jazz 6. Escalas y entonación de escalas	1.Reconocimiento de compases 2.Reconocimiento de pulsos 3.Comparación de ritmos 4. Dictado rítmico 5. Elementos rítmicos 6. Dictado de elementos rítmicos 7.Imitación rítmica 8. Estilos rítmicos	1. Oído absoluto 2. Entonación contrapuntística 3. Comparación melódica 4. Dictados melódicos 5. Reconocimiento de notas 6. Comparación de alturas 7. Dictado de alturas 8. Imitación de alturas 9. Lectura a primera vista 10. Afinación	1. Comparación de acordes 2. Imitación de acordes 3. Reconocimiento de acordes 4.Entonación de acordes 5. Cluster de acordes 6. Imitación de acordes de Jazz 7. Acordes de Jazz 8. Entonación de acordes de Jazz	1. Progresiones armónicas avanzadas 2. Cadencias 3. Progresiones de acordes 4. Formas 5. Formas del Jazz 6. Progresiones de Jazz 7. Modulación

Auralia ofrece otras funciones como creación de pruebas de evaluación y almacenamiento de los puntajes del usuario, además un libro de texto titulado *The*

[30] http://www.risingsoftware.com/auralia30/

performing ear[31], el cual es un método de entrenamiento auditivo que se basa en el uso de Auralia.

De todas las funciones de Auralia solamente la sección *Cluster de acordes* ofrece un ligero acercamiento a la música no tonal.

EarMaster 5.0 (sólo para sistema operativo Windows)[32]

EarMaster es un programa desarrollado por la empresa danesa EarMaster Aps. Su primera versión salió en 1994. Actualmente cuenta con dos versiones 5.0: EarMaster School y EarMaster Pro. Ambas versiones estan organizadas en 12 secciones, como a continuación se presenta:

Secciones del programa EarMaster 5.0
1. Entonación de intervalos
2. Identificación de intervalos
3. Comparación de intervalos
4. Identificación de escalas
5. Identificación de acordes
6. Identificación de inversiones de acordes
7. Identificación de progresiones de acordes
8. Dictados rítmicos
9. Lectura de ritmos
10. Imitación rítmica
11. Corrección rítmica
12. Dictados melódicos

EarMaster tampoco ofrece algún apartado para la música no tonal.

[31] http://www.rising.com.au/auralia21/performing_ear.shtml

[32] http://www.earmaster.com

Auralia y EarMaster son dos ejemplos típicos de las posibilidades y alcances de los programas de cómputo para el entrenamiento auditivo que se encuentran disponibles actualmente.

2.4 Generación de material musical a través de la computadora

Aparte de carecer de un estudio de la música no tonal, las aplicaciones de cómputo para el entrenamiento auditivo carecen también de material didáctico vasto. El problema de suficiencia en el material didáctico de trabajo (específicamente las melodías y progresiones armónico-rítmicas) de las aplicaciones de cómputo radica principalmente en la forma en que se obtiene el material musical. Un método es la creación de bases de datos. La desventaja de éste es que ofrece material de estudio en cantidad limitada. El otro método es la generación automática del material musical. Este método, si bien es más complejo, es capaz de proporcionar material diverso e ilimitado.

2.4.1 *Hacia una Teoría computacional general de la estructura musical (Cambouropulos* 1998*)*

Desarrollar métodos de generación de material musical requiere un análisis de la estructura musical, el cual es traducido posteriormente a algoritmos de programación. En la Tesis doctoral *Toward a General Computational Theory of Musical Structure* (*Hacia una teoría computacional general de la Estructura Musical*)[33] Emilios Cambouropulos esboza una *Teoría computacional general de la estructura musical (GCTMS,* por sus siglas en inglés), cuyo objetivo es obtener la descripción estructural en términos computacionales de la superficie musical de una obra; independientemente del estilo de que se trate. Al respecto señala Cambouropulos:

> La obra musical es presentada a la *GCTMS* como una secuencia de eventos musicales discontinuos simbólicamente representados (i.e. notas) sin elementos estructurales de alto nivel (i.e. articulaciones escritas por el compositor o el ejecutante, indicaciones de tempo o carácter, etc.); no obstante que dicha información podría ser usada de manera constructiva para guiar el proceso analítico.[34]

[33] Cambouropulos Emilios, *Toward a General Computational Theory of Musical Structure*, The University of Edinburgh, Department of Artificial Intelligence, Ph.D. (Tesis Doctoral), Edinburgh 1998, página 3 (Traducción del autor).

[34] Ibidem página 3.

La teoría de Cambouropulos establece dos niveles de desarrollo:

1. Desarrollo de un número de componentes individuales que se centran en tareas musicales analíticas especializadas. Lo cual incluye:

1.1 Representación general de alturas-intervalos y su transcripción algorítmica

1.2 Modelos de detección de las fronteras locales

1.3 Modelos de acentuación y estructura métrica

1.4 Algoritmo de inducción de patrones de cadenas y función de selección

1.5 Algoritmo de decodificación y formación de categorías

2. Desarrollo de un listado de métodos que describan cómo estos componentes se relacionan e interactúan con el fin de arribar a una descripción plausible estructural de una superficie musical.[35]

Los métodos señalados por Cambouropulos en el punto dos son aquellos que se refieren a la generación del material musical. En otras palabras, los modos de relación e interacción entre los sonidos van a determinar las diferentes formas de generación de material musical con la computadora.

[35] Ibidem pag. 4.

2.4.2 Principales métodos de generación de material musical con la computadora

Para el compositor Eduardo Reck Miranda[36] la generación de material musical se mueve entre dos tendencias básicas: el formalismo radical (p.e. el serialismo radical) el cual establece reglas muy estrictas y poco permisivas y la aleatoriedad absoluta. Un método intermedio de generación de material musical es el uso de la estadística y las probabilidades «con el fin de hacer predicciones generales concernientes a un conjunto de fluctuaciones aleatorias».[37] Xenakis, por ejemplo, utiliza este tipo de métodos en la generación de material musical.

En su libro *Composing Music with Computers*, Eduardo Reck Miranda presenta los principales métodos de generación de material musical con la computadora.

- Probabilidades
- Gramáticas
- Autómatas
- Algoritmos iterativos: Caos y fractales
- Modelos neuronales

2.4.2.1 Probabilidades

Si bien la generación de material musical puede ser completamente estocástica normalmente se prefieren los procesos cuyo resultado está influenciado por resultados o parametrizaciones previas. Estos resultados son conocidos como *probabilidades condicionadas*.[38] Señala Miranda:

> Por ejemplo, con el fin de generar una melodía a partir de un conjunto de notas, la computadora puede ser programada para, a través de un método aleatorio, seleccionar una nota a la vez para posteriormente tocarla con un sintetizador. Si no hay notas duplicadas en el conjunto, todas las notas tendrán la misma probabilidad de ser escogidas. Al contrario si existe mas de una nota duplicada, la posibilidad de que ésta sea seleccionada aumentará. Intuitivamente, así es como las probabilidades

[36] Reck Miranda Eduardo, *Composing Music with Computers*, Focal Press, Oxford 2001, pag. 57-58 (Traducción del autor).

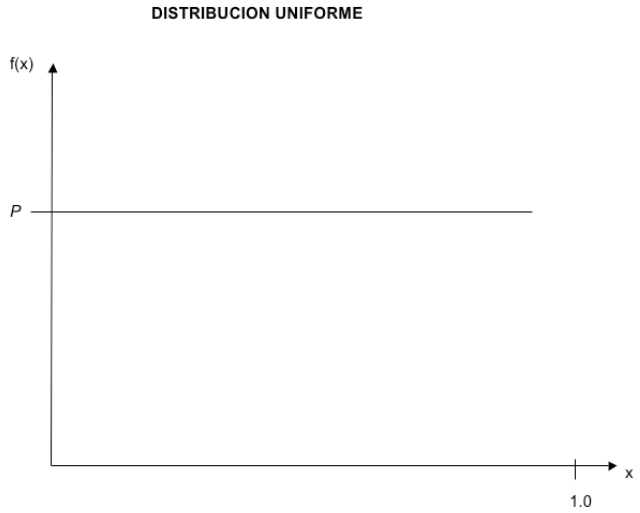
[37] Ibidem pag. 58.

[38] Ibidem pag. 62.

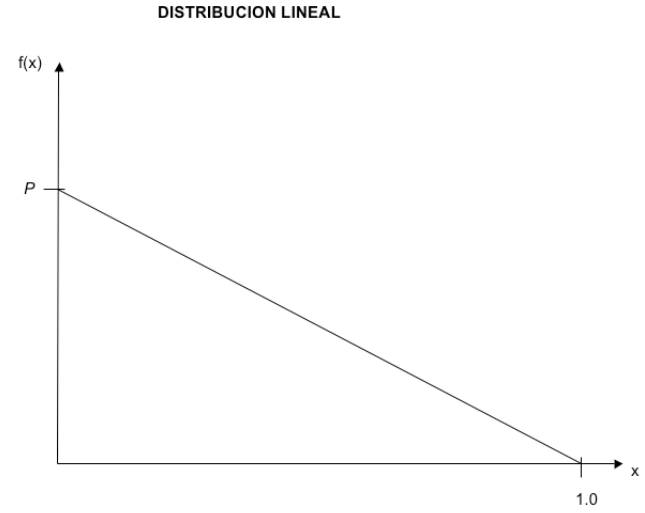
operan: *al repetir notas en un conjunto las oportunidades de que esas notas sean escogidas se incrementará proporcionalmente respecto al número de repeticiones.*[39]

Una forma de parametrización es el empleo de funciones de distribución:[40]

1. *Distribución uniforme*. Todos los miembros del conjunto tienen la misma posibilidad de ser seleccionados.



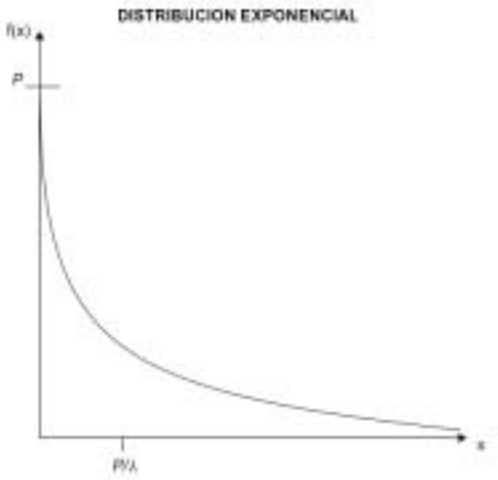
2. *Distribución lineal*. En este caso los valores que se encuentran cerca del lado izquierdo del eje horizontal de la gráfica poseen mayor probabilidades de selección que aquellos que se encuentren del lado derecho.



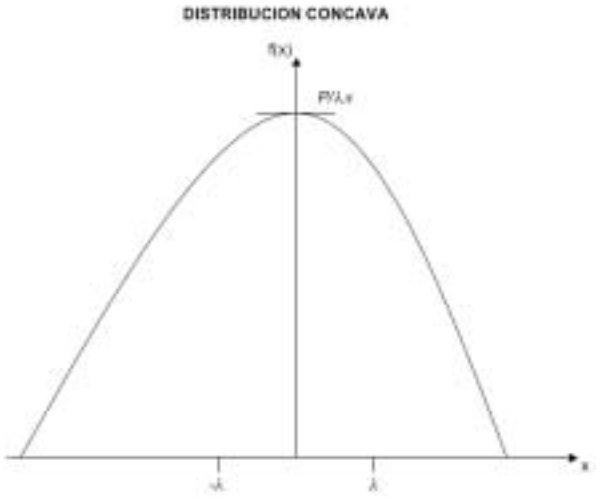
[39] Ibidem pag. 62.

[40] Ibidem pag. 63.

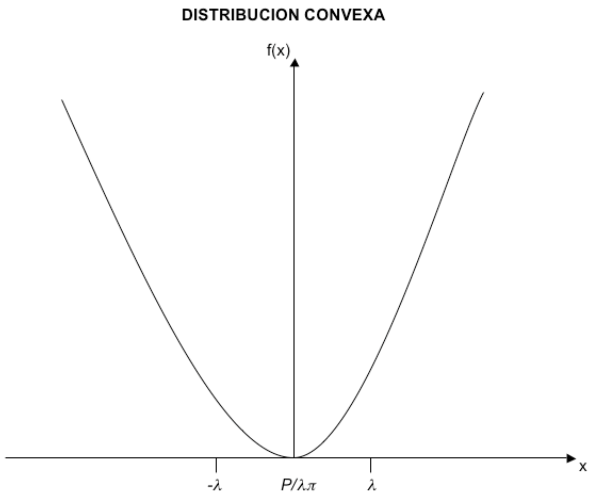
3. *Distribución exponencial*. Al igual que la distribución lineal, la distribución exponencial favorece los valores cercanos al lado izquierdo del eje horizontal de la gráfica; sin embargo existe un valor λ que controla el grado de "favoritismo". Entre mayor sea el valor de λ mayor será la tendencia a obtener valores más cercanos al lado izquierdo de la curva.



4. *Distribución cóncava*. Ésta posee una distribución bilateral. Su función de densidad tiene la forma de una curva cóncava y depende también del parámetro λ, donde entre mayor sea el valor de λ, más amplia resulta la curva.



5. *Distribución convexa*. La distribución convexa posee la forma inversa de la distribución cóncava, en el sentido de que los valores con un mayor índice de probabilidad son puntos finales.



Otros métodos de generación de material musical con el uso de la probabilidad y la estadística, señalados por Miranda, son las tablas de probabilidades, como las cadenas de Markov.[41]

2.4.2.2 Gramáticas

Las gramáticas se refieren a un cierto tipo de organización jerárquico-estructural. En su libro titulado *Syntactic Structures (Estructuras sintácticas)*[42] Noam Chomsky «sugiere que los humanos son capaces de hablar y entender un lenguaje mayoritariamente porque tenemos la habilidad de dominar su gramática.»[43] Chomsky cree que es posible definir una gramática universal, aplicable a cualquier lenguaje; a través de un formalismo matemático que describa completamente su funcionamiento: reglas formales de descripción, generación y transformación de oraciones.

Un ejemplo de aplicación de las gramáticas formales en la generación de material musical es el programa de composición musical EMI (*Experiments in Musical Intelligence*) de David Cope.[44] Este programa utiliza una técnica de escritura de gramáticas formales denominada *Red de transición Avanzada* (ATN por sus siglas en inglés). EMI cuenta con

[41] Ibidem pag. 63.

[42] Chomsky Noam, *Syntactic Structures*, Walter de Gruyter, GmbH, Berlin 1957.

[43] Reck Miranda Eduardo, *Composing Music with Computers*, Focal Press, Oxford 2001, pag. 74.

[44] Cope David, *Virtual Music*, The MIT Press, Cambridge Massachusetts 2001.

una base de datos la cual es cargada con ejemplos musicales del estilo musical con el que se desea que genere material musical. El sistema explora dicha base de datos buscando estadísticamente aquellos pasajes que indiquen el estilo de la pieza, a la vez de definir las funciones gramaticales, para posteriormente generar material musical nuevo.

2.4.2.3 Autómatas

Los autómatas funcionan de manera similar a las gramáticas formales. De hecho, los *Autómatas de Estado Finito* (FSA por sus siglas en inglés) son herramientas en el diseño de lenguajes de programación. Apunta Miranda:

> Los autómatas de estado finito son empleados por los programadores para especificar leyes de formación de cadenas para construir *parsers* [analizadores sintácticos] para los compiladores o intérpretes de los lenguajes de cómputo…Los autómatas suelen ser más eficientes que las gramáticas formales en estructuras de bajo nivel o pasajes musicales cortos.[45]

2.4.2.4 Algoritmos iterativos

> Un proceso iterativo es la aplicación repetida de un procedimiento matemático donde cada paso es aplicado a la salida del paso precedente [la salida del proceso es retroalimentada a la entrada del mismo]. Matemáticamente, un proceso iterativo es definido como una regla que describe una acción que deberá ser aplicada repetidamente a un valor inicial de X_0. El resultado de un proceso iterativo constituye un conjunto, técnicamente señalado como la *órbita* del proceso; los valores de este conjunto son conocidos como los *puntos* del la órbita.[46]

Un proceso iterativo puede producir tres clases de órbitas:

1. Orbitas cuyos puntos tienden hacia un valor fijo y estable.
2. Orbitas cuyos puntos tienden a oscilar entre elementos específicos.
3. Orbitas cuyos puntos caen en el caos.

---

[45] Reck Miranda Eduardo, *Composing Music with Computers*, Focal Press, Oxford 2001, pag. 79 (Traducción del autor).

[46] Ibidem pag. 83.

Si bien, según Miranda, «no existen aún reglas claras para la construcción de formas musicales provenientes del caos»[47], son las órbitas cuyos puntos tienden al caos las más propicias para la generación de material musical; esto ya que las órbitas que tienden a estabilizarse en uno o varios puntos producen resultados musicales menos interesantes.

2.4.2.5 Modelos Neuronales

Los modelos neuronales tienden a imitar la manera como funciona el cerebro; específicamente a partir de la creación de redes neuronales. El principio básico de operación de las redes neuronales en la teoría de la computación consiste en «proveer mecanismos de aprendizaje necesarios para que el sistema de cómputo pueda ser programado a partir de la simple y repetida exposición de la red al comportamiento deseado.»[48] En un modelo neuronal el proceso de aprendizaje de la red se da continua e ininterrumpidamente.

Tabla: Algunos programas de cómputo generadores de material musical[49]

Programas	**Funcionamiento**
Nyquist	Composición algorítmica en lenguaje LISP
OpenMusic	Programación visual en lenguaje LISP
Bol Processor	Composición a través de gramáticas formales
Textura	Música generada aleatoriamente y por probabilidades
MusiNum	Generación de notas MIDI a través de series de números
MusicGenerator y Fractmus	Generadores de música a través de algoritmos iterativos
CAMUS	Composición a través de autómata celular

[47] Ibidem pag. 89.

[48] Reck Miranda Eduardo, *Composing Music with Computers*, Focal Press, Oxford 2001, pag. 103.

[49] Reck Miranda Eduardo, *Composing Music with Computers*, páginas 177-204.

La aplicación de cómputo *ModusXXI* esta desarrollada a partir de un método probabilístico de principios formales de generación melódica. En *ModusXXI* se establecen ciertas reglas, como la selección del material interválico (proveniente de la metodología *Modus Novus*), las figuras rítmicas, la métrica, etc.; creando subconjuntos del conjunto total de parámetros modificables. Una vez establecidos estos subconjuntos, es aplicado un método aleatorio que lleva a cabo un proceso de selección con una distribución probabilística uniforme. En otras palabras, una vez seleccionados los miembros del subconjunto, van a tener todos la misma probabilidad para ser escogidos.

Los aspectos sobre la implementación, desarrollo y funcionamiento de *ModusXXI* serán expuestos en los capítulos subsecuentes.

3. Desarrollo

3.1 Teoría de programación e ingeniería de software

3.1.1 Fases de desarrollo de un programa de cómputo

En su libro titulado *Foundations of Computer Science*, Behrouz A. Forouzan señala: «el proceso de desarrollo en el ciclo de vida del software implica cuatro fases: análisis, diseño, implementación y prueba.»[1]

1. Análisis. Esta etapa consiste en determinar cuál es la tarea que el programa debe cumplir. El análisis implica cuatro pasos:
   - 1.1 Definición del usuario. Un programa de cómputo puede estar diseñado para un usuario genérico o un usuario específico. Éste debe estar claramente definido.
   - 1.2 Definición de las necesidades. El usuario debe definir claramente cuáles son las necesidades que desea cubrir con el programa.
   - 1.3 Definición de los requisitos. Basado en las necesidades del usuario, el analista puede definir con exactitud los requisitos del sistema a desarrollar.
   - 1.4 Definición de los métodos. Una vez definidos los requisitos, el analista puede seleccionar los métodos necesarios para resolver los requisitos.

2. Diseño. La fase de diseño define "cómo" se va a llevar a cabo "aquello" que fue definido en la fase de análisis.
   - 2.1 Modularidad. El paquete de cómputo a desarrollar es dividido en pequeños módulos. Cada módulo es diseñado, probado y relacionado con otros módulos a través del programa principal.
   - 2.2 Herramientas. La fase de diseño se vale de muchas herramientas, entre ellas los diagramas de estructura (*structure chart*). El diagrama de flujo muestra la interacción entre las parte (módulos).

3. Implementación. En esta fase es creado el programa.
   - 3.1 Herramientas. Las herramientas más usuales en esta fase son los diagramas de flujo y el pseudocódigo. Los diagramas de flujo usan símbolos gráficos

[1] Forouzan Behrouz A., *Foundations of Computer Science*, Thomson Books/Cole, 2003 Canada, páginas 196-198 (T del A).

para representar el flujo de datos a través de los módulos. El pseudocódigo es parte texto explicativo y parte lógica de programación que describe, en detalles algorítmicos precisos, lo que el programa va a realizar.

3.2 Codificación. Después de hacer los diagramas de flujo y/o el pseudocódigo el programador escribe el código en un lenguaje específico. La elección del lenguaje se basa en la eficiencia del lenguaje para la aplicación en particular.

4. Prueba. Una vez que el programa ha sido escrito, debe ser probado. Existen dos modelos de prueba: prueba de caja negra (*Black Box Testing*) y prueba de caja blanca (*White Box Testing*).

4.1 Prueba de caja negra. Esta prueba es realizada por el ingeniero de prueba del sistema y el usuario, sin saber qué es lo que pasa dentro del programa.

4.2 Prueba de caja blanca. Es realizado por el programador, el cual debe saber qué es exactamente lo que sucede dentro del programa.

3.1.2 Modelos de procesos de desarrollo

Existen básicamente dos tipos de modelos de procesos de desarrollo: el *modelo de cascada* y el *modelo incremental*.[2]

1. Modelo de cascada (*Waterfall model*). En este modelo el proceso de desarrollo se mueve en una sola dirección. Esto quiere decir que una fase de desarrollo no puede iniciarse hasta que no haya sido concluida la anterior. Por ejemplo, la fase de desarrollo no puede iniciarse hasta que no sea concluida la de análisis.
2. Modelo incremental. Aquí el proceso es desarrollado en una serie de pasos. Primero se completa una versión simplificada del programa. Esta versión representa el conjunto completo, pero no incluye detalles.

[2] Ibidem páginas 198-199.

3.1.3 Calidad

Forouzan define al software de calidad como:

> Aquel que satisface los requisitos explícitos e implícitos del usuario, está bien documentado, complace los estándares operativos de la organización que lo requiere, y corre eficientemente en el hardware para el cual fue desarrollado.[3]

Forouzan establece los siguientes factores de calidad en el software:[4]

1. Operabilidad. La operabilidad se refiere a las operaciones básicas de un sistema e involucra los siguientes factores:
   - 1.1 Precisión. La precisión de un programa puede ser medida en relación al número de fallos, errores de programación y solicitudes de cambios por parte del usuario.
   - 1.2 Eficiencia. La eficiencia puede medirse como la respuesta en tiempo real durante la ejecución; la cual debe ser recibida dentro de un segundo en el 95% de las veces.
   - 1.3 Confiabilidad. Esta es la suma de los otros factores.
   - 1.4 Seguridad. Es la facilidad con que usuarios no autorizados pueden acceder los datos del sistema.
   - 1.5 Respuesta oportuna (*Timeliness*). Es el tiempo requerido para el almacenamiento y procesamiento de datos.
   - 1.6 Facilidad de Uso. Es la sencillez y simplicidad en el manejo del programa.

2. Mantenimiento. Este aspecto se refiere a mantener al sistema corriendo correctamente y actualizado.
   - 2.1 Capacidad de cambio y flexibilidad. Facilidad para hacer cambios o alteraciones en el sistema.
   - 2.2 Funcionamiento correcto. La prevención y detección de errores propicia el funcionamiento correcto de la aplicación.
   - 2.3 Capacidad de prueba. Facilidad de evaluación del programa.

[3] Ibidem. página 202.

[4] Ibidem páginas 203-205.

3. Capacidad de transferencia. Se refiere a la posibilidad de mover datos y/o sistemas de una plataforma a otra para reutilizar el código.
   3.1 Reutilización del código. Creación de bibliotecas de código que pueda ser reutilizable.
   3.2 Interoperabilidad. Es la capacidad de enviar datos a otros sistemas.
   3.3 Portabilidad. Es la habilidad de mover software de una plataforma de hardware a otra.

3.1.4 Documentación

Si se desea usar correctamente y mantener eficiente a un programa de cómputo es necesario contar con documentación. Forouzan establece dos tipos básicos de documentación: la documentación de usuario y la documentación del sistema.[5]

1. Documentación de usuario. Normalmente es el manual.
2. Documentación del sistema. Esta documentación define al paquete de cómputo en sí. Debe ser escrito para que se le pueda dar mantenimiento y pueda ser modificado el programa por otros programadores.

3.2 Teorías de desarrollo de interfaces de usuario

En su libro titulado *Diseño de interfaces de usuario*[6] Ben Shneideman establece tres grupos generales de teorías, de acuerdo al criterio de estudio del desarrollo de interfaces. Estos grupos son: teorías de análisis o descriptivas, teorías a nivel de componentes de interfaz (*widgets*) y teorías de contexto de uso.

3.2.1 Teorías de análisis o descriptivas:

Este grupo de teorías comprende cuatro modelos: modelos de niveles, modelos de etapas de acción, GOMS y el modelo de nivel de pulsación.

---

[5] Ibidem. página 206.

[6] Shneiderman Ben, *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (Trad. al Español, Jesús Sánchez Cuadrado), Pearson Educación, Madrid 2006, páginas 99-117.

Modelo de niveles

Un modelo de este tipo de teorías es el esquema secuencial de cuatro niveles o en cascada. Dichos niveles son:

1. *Conceptual*. Es el modelo mental del usuario. Las decisiones sobre el nivel conceptual afectan a los otros niveles.
2. *Semántico*. Describe el significado asociado a la entrada de datos por parte del usuario y a la visualización de salida de la computadora. Por ejemplo, dos métodos distintos pueden efectuar una misma tarea, en este caso si bien los procesos son distintos el resultado es el mismo.
3. *Sintáctico.* Define cómo las acciones del usuario, con una intención semántica, conforma sentencias completas que dan instrucciones a la computadora para realizar determinadas acciones.
4. *Léxico*. Se encarga de las dependencias entre dispositivos y de los mecanismos precisos con los que los usuarios especifican la sintaxis.

Modelo de etapas de acción

Este modelo identifica siete etapas de acción, organizadas en un patrón cíclico, como modelo explicativo de la interacción humano-máquina:

1. Formulación del objetivo
2. Formulación de la intención
3. Especificación de la acción
4. Ejecución de la acción
5. Interpretación del estado del sistema
6. Evaluación del resultado

Asimismo, este modelo sugiere cuatro principios para un buen diseño:

1. El estado y las alternativas de acción deben ser visibles
2. El modelo conceptual y el sistema deben ser consistentes.
3. La interfaz debe poseer correspondencias que pongan de manifiesto las relaciones entre etapas.

4. Los usuarios deben ser retroalimentados continuamente.

   Este modelo es muy útil para describir la exploración del usuario a través la interfaz.

GOMS y el modelo de nivel de pulsación

El modelo *objetivos, operadores, métodos y reglas de selección* (***G****oals*, ***O****perators*, ***M****ethods and* ***S****election rules*) postula que los usuarios comienzan formulando objetivos, luego piensan en términos de operadores (actos elementales cuya ejecución es necesaria para cambiar el estado actual de algo) y utiliza procedimientos basados en reglas de selección; las cuales son estructuras de control que sirven para elegir métodos para lograr un objetivo.

El modelo de nivel de pulsación es una versión simplificada del GOMS que además predice los tiempos de ejecución de tareas.

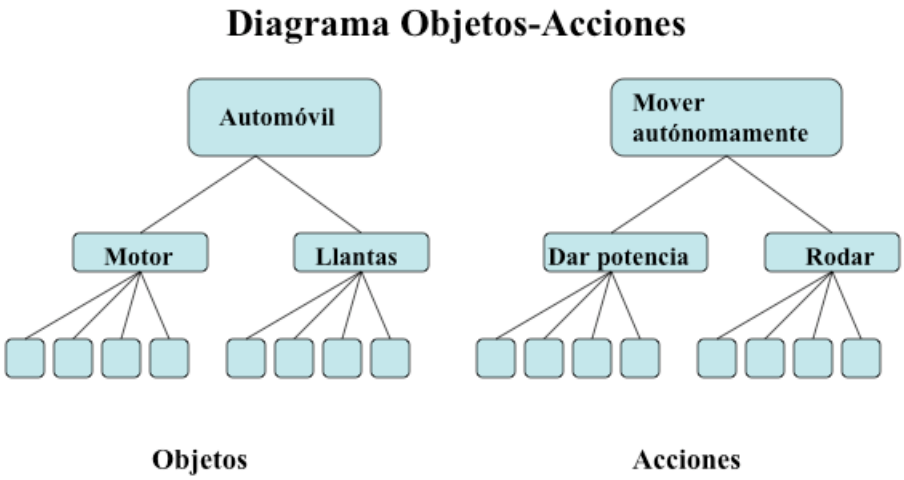
3.2.2 Teorías a nivel *widget* (componentes de la interfaz)

Una forma distinta de acercamiento al desarrollo de interfaces de usuario es siguiendo las simplificaciones que ofrecen las herramientas de creación de más alto nivel. En este caso, se crea un modelo a partir de los componentes (*widgets*) que soporta la interfaz. Por ejemplo, se analiza el comportamiento del usuario con un componente tipo lista con barra de desplazamiento. Esto puede servir para predecir el comportamiento de futuros usuarios con dicho componente; así como para mejorar el diseño del mismo.

Modelo de interfaz objeto-acción

El modelo OAI (*Object-Action-Interface*) es un modelo descriptivo y explicativo que se centra en objetos y acciones de tarea e interfaz. Los detalles sintácticos son mínimos, por tanto los usuarios que conocen los objetos y acciones del dominio de tarea pueden aprender la interfaz de forma relativamente sencilla. El modelo OAI refleja el diseño de más alto nivel, con el que tratan muchos diseñadores cuando usan *widgets* en las herramientas de construcción de interfaces de usuario. Los *widgets* estándar tienen una sintaxis simple y formas simples de realimentación. El modelo OAI está en armonía con el método de ingeniería de software del diseño orientado a objetos.

Hacer un diseño objeto-acción comienza con la comprensión de la tarea. Esa tarea incluye el universo de objetos con los que trabaja el usuario y las acciones que aplican a esos objetos. Los objetos de tarea de alto nivel son los productos finales que desea el usuario. Estos pueden descomponerse en objetos intermedios y finalmente en unidades atómicas. Asimismo, las acciones de tarea comienzan con intenciones de alto nivel que se descomponen en tareas intermedias y tareas específicas.



3.2.3 Teorías de contexto de uso

Estos modelos enfatizan en el entorno social, en la motivación de los usuarios o en la experiencia, estudiando el proceso de transición de principiante a experto, las diferencias en los niveles de conocimiento, así como la observación etnográfica. En estos modelos los usuarios son participantes en el proceso de diseño.

3.2.4 Usabilidad universal: paradigma del desarrollo de interfaces de usuario

Ben Shneideman[7] establece tres características inherentes a la calidad en el desarrollo de interfaces de usuario: *usabilidad, universalidad y utilidad*.[8] La *utilidad* es la capacidad de la interfaz para facilitar el éxito del usuario para resolver una tarea propuesta,

[7] Shneiderman Ben, *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (Trad. al Español, Jesús Sánchez Cuadrado), Pearson Educación, Madrid 2006 .

[8] Ibidem pag. 12.

la *universalidad* es la capacidad de la interfaz que permite a usuarios de distintos contextos, pueden ser sociales, culturales, de habilidades, etc., realizar exitosamente una tarea propuesta. Finalmente la *usabilidad* es la facilidad de manejo que ofrece la interfaz. En este sentido, tanto la *utilidad* como la *universalidad* son capacidades que para Shneideman se sintetizan en lo que denomina la *usabilidad universal*.[9]

El primer objetivo de la *usabilidad* debe ser el análisis de requisitos. Shneideman nos presenta los siguientes puntos a considerar para el análisis de requisitos:[10]

1. *Determinar las necesidades del usuario.* ¿Qué tareas y subtareas debe realizar el usuario? (Puede ser peligroso proporcionar una funcionalidad excesiva.)
2. *Asegurar una fiabilidad apropiada.* Las acciones deben funcionar como se especificó, los datos deben reflejar el contenido de las bases de datos, y las actualizaciones se deben realizar correctamente.
3. *Estimular la estandarización, integración, consistencia y portabilidad adecuadas.* La estandarización hace referencia a características comunes de la interfaz de usuario a través de muchas aplicaciones. La consistencia se refiere a secuencias de acciones, términos, unidades, composiciones, tipografía, etc. que son comunes dentro de la aplicación. La integración es una resultante de la consistencia. La portabilidad hace referencia al potencial de convertir datos y compartir interfaces de usuario entre múltiples entornos de software y hardware.
4. *Completar los proyectos a tiempo y dentro de presupuesto.*

El éxito de la usabilidad puede ser medido a través de los siguientes parámetros:[11]

1. *Tiempo de aprendizaje.* ¿Cuánto tiempo requiere el usuario para aprender a manejar una aplicación?
2. *Velocidad de realización de tareas.* ¿Cuánto tiempo tarda en realizar exitosamente una tarea el usuario?

[9] Ibidem pag. 26.

[10] Ibidem pag 14.

[11] Ibidem pag. 17.

3. *Porcentajes de errores de los usuarios.* ¿Cuántos y qué clase de errores comete el usuario al realizar sus tareas?
4. *Retención de conocimientos del usuario respecto a la aplicación con el paso del tiempo*. ¿Mantiene el usuario sus conocimientos después de una hora, un día, una semana?
5. *Satisfacción subjetiva.* ¿Le gustó al usuario usar diferentes aspectos de la interfaz?

La usabilidad universal, como se mencionó arriba, engloba otras características de calidad en el desarrollo de interfaces de usuario. En este sentido Shneideman sugiere tomar en cuenta los siguientes aspectos a la hora de desarrollar una interfaz que busque la usabilidad universal:[12]

1. *Variación en capacidades físicas y de lugares de trabajo*. Por ejemplo grupos de discapacitados, así como los sectores urbanos y rurales.
2. *Diversidad de capacidades cognitivas y perceptivas.* También en esta clasificación pueden entrar grupos de discapacitados, pero también los niños y los adultos mayores.
3. *Diferencia de personalidad.* Aquí pueden reconocerse cuatro tipos básicos de dicotomías en personalidades:
    - *Extroversión vs. Introversión*
    - *Sensación vs. Intuición*
    - *Percepción vs. Juicio*
    - *Sentimiento vs. Pensamiento*
4. *Diversidad cultural e internacional*. En este caso son evaluadas las diferencias que tienen que ver con el trasfondo cultural, étnico, racial o lingüístico.

Considerar estos aspectos en el desarrollo de interfaces tiene consecuencias positivas no sólo en el sector de usuarios al que se está dirigiendo el desarrollo de la interfaz, sino también en usuario más expertos o de capacidades de uso mayores. Por ejemplo, una interfaz que ofrece funciones de mensajes en pantalla para débiles visuales puede ser también útil para usuarios sin esta discapacidad.

[12] Ibidem páginas 27-35.

3.2.5 Principios para el desarrollo de interfaces de usuario

Shneiderman establece los siguientes principios en el desarrollo de interfaces de usuario:[13]

a) *Determinar el nivel de competencia de los usuarios*. Podemos distinguir entre:
   - Usuarios principiantes o nuevos
   - Usuarios ocasionales con conocimientos
   - Usuarios habituales expertos

   No obstante, el diseño de una interfaz puede abarcar varios de estos grupos. En este caso se requiere de un diseño de aproximación *multi-capa*, también llamado *estructura en niveles* o en *espiral*. En este tipo de diseños el aprendizaje es gradual y consecuentemente pueden acceder tanto usuarios novicios como expertos.

b) *Identificar las tareas*. Es importante saber tanto las tareas como las acciones de los usuarios y clasificar su frecuencia, de tal forma que el usuario no pueda ejecutar tareas innecesarias las cuales puedan causar desordenes inoportunos.

c) *Elegir el estilo de interacción*. Existen básicamente cinco estilos de interacción:
   - *Manipulación directa*. Aquí son representados de manera visual los objetos y acciones, de tal forma que el usuario interactúa directamente con ellos. Este método resulta muy atractivo para los principiantes.
   - *Selección de menú*. El usuario lee una lista de elementos, selecciona el más apropiado para su tarea y observa sus efectos.
   - *Formularios.* El usuario introduce datos en campos específicos para su posterior procesamiento.
   - *Lenguaje de órdenes.* Son el dominio de usuarios expertos hábiles. Requieren el dominio de un conjunto complejo de semántica y sintaxis.
   - *Lenguaje natural.* Este tipo de interacción está aún en desarrollo y puede resultar lento e impredecible.

d) *"Ocho reglas de oro de la programación"*:
   - *Esforzarse por conseguir consistencia.* Estandarización de terminología, procesos, etc. comunes en la aplicación.

[13] Ibidem páginas 76-94.

- *Atender a la usabilidad universal*. Ofrecer una interfaz capaz de interactuar con usuarios novatos y expertos, así como de distintos entornos.
- *Ofrecer realimentación informativa*. Por cada acción del sistema debiera darse una reacción del sistema mismo que permita al usuario su identificación.
- *Diseñar diálogos que conduzcan a la finalización de la tarea*. Las secuencias de acción deben organizarse en grupos con comienzo, mitad y final.
- *Prevenir errores*. Prevenir errores es uno de los aspectos más importantes en el desarrollo de interfaces de usuario. El diseño del sistema debe estar hecho para que el usuario cometa la menor cantidad de errores. Si un usuario comete un error, la interfaz debe localizarlo y ofrecer instrucciones de recuperación simples, consecutivas y concretas. Igualmente, los errores no debieran alterar el estado del sistema, y en su defecto dar instrucciones de cómo restaurarlo.
- *Permitir deshacer acciones de forma fácil*. En la medida de lo posible las acciones deben ser reversibles.
- *Dar soporte al control del usuario*. El usuario necesita sentir que está al mando de la interfaz y que ésta responde a sus acciones.
- *Reducir la carga de la memoria a corto plazo*. Las visualizaciones deben mantenerse simples y el tiempo de entrenamiento para aprender códigos, nemotecnias, y secuencias debe ser suficiente.

e) *Integrar la automatización manteniendo el control humano*. Dado que el mundo real es un *sistema abierto*, el papel humano de supervisión debe mantenerse. Los diseños deben otorgar al usuario el control y automatizar sólo para mejorar el rendimiento del sistema.

3.2.6 Manuales de usuario

Shneiderman[14] distingue entre dos clases principales de manuales de usuario: *manuales en papel y manuales en línea* (digitales).

Entre los manuales en papel se encuentran las siguientes categorías:

- *Manual de instalación*. Instrucciones paso por paso para configurar la aplicación.
- *Notas breves de inicio rápido*. Instrucciones que permiten a usuarios impacientes probar por primera vez las características de la aplicación.
- *Tutorial introductorio.* Explicaciones básicas de la aplicación con ejemplos.
- *Tutorial exhaustivo*. Explicaciones que cubren tareas típicas y avanzadas.
- *Manual de referencia detallado*. Cubre la totalidad de las características.
- *Referencia rápida*. Presentación concisa de la sintaxis de la aplicación.
- *Manual de migración*. Presentación de las características del sistema a usuarios que poseen conocimientos de un sistema parecido o de versiones previas del mismo.

Si bien muchos de los manuales en papel se encuentran en línea es posible reconocer las siguientes variaciones en línea:

- *Manual en línea*. A diferencia del formato en papel, los manuales en línea hacen que el texto esté disponible más fácilmente y ofrecen opciones de búsqueda y se pueden actualizar más fácilmente.
- *Ayuda en línea*. Descripción breve y solución a posibles problemas que se le puedan presentar al usuario. Puede proporcionar índice de términos, búsqueda por palabras, consejos paso a paso y acceso a información complementaria en Internet.
- *Ayuda sensible al contexto*. Ayuda interactiva controlada por el usuario. Va desde recuadros explicativos sobre algún objeto hasta asistentes iniciados por el sistema que registran las actividades de los usuarios y proporcionan información relevante.
- *Tutorial en línea*. Entorno de formación en línea que usa medios electrónicos para enseñar a los principiantes. Puede variar desde presentaciones de dos minutos hasta cursos de formación por computadora de una semana.

---

[14] Shneiderman Ben, *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (Trad. al Español, Jesús Sánchez Cuadrado), Pearson Educación, Madrid 2006, páginas 595-596.

- *Demostración animada*. Exposición de diapositivas, secuencias de capturas de pantallas o secuencias de video con explicaciones verbales.
- *Guías*. Grabaciones de audio o video de personas o personajes animados que ofrecen introducciones o lecciones concretas que cubren temas clave.

Otros métodos son archivos de preguntas de usuarios o de diálogos entre usuarios:

- *FAQs* (*Frequently Asked questions*, Preguntas más frecuentes). Son respuestas preparadas por los desarrolladores a preguntas habituales de los usuarios.
- *Comunidades en línea, grupos de noticias, listas de correo electrónico, Chat, y mensajería instantánea*. Ofrecen respuestas a los usuarios, por parte de otros usuarios a preguntas concretas.

3.3 Desarrollo de la aplicación de cómputo *ModusXXI*

3.2.1 Análisis

Definición del usuario. *ModusXXI* es un programa diseñado para el estudio de la música no tonal en la clase de adiestramiento auditivo. Son sus usuarios los estudiantes de música que desean ejercer una actividad musical profesional, así como los profesores que deseen incorporar esta aplicación de cómputo dentro de su plan de estudios. Un grupo secundario de usuarios pueden ser los compositores que deseen generar material musical a través de un sistema melódico interválico no tonal.

Definición de necesidades. Actualmente, los estudiantes y profesores de la clase de adiestramiento auditivo requieren herramientas que faciliten la realización de dictados y la generación de ejemplos melódicos para la lectura; específicamente de música no tonal. La aplicación de cómputo *ModusXXI* genera un número ilimitado de melodías no tonales. Con *ModusXXI* tanto el estudiante como el maestro pueden generar material melódico no tonal, suficiente para el estudio del *Modus Novus*, o alguna otra metodología de adiestramiento auditivo no tonal.

Definición de requisitos. La aplicación de cómputo debe ser capaz de crear un número ilimitado de melodías que sirvan para el dictado y la lectura de música no tonal. El programa de cómputo *ModusXXI* debe ofrecer los siguientes aspectos al usuario:

1. La generación de una melodía, de acuerdo a las características musicales asignadas.
2. Un reproductor de la melodía (audio).
3. Un archivo gráfico con la representación en partitura de la melodía creada por el programa *ModusXXI*; la cual pueda ser cotejada por el usuario, luego de un dictado basado en la versión audible.
4. Una interfaz gráfica con una estructura de control que facilite al usuario la selección de características musicales para la generación de la melodía; por ejemplo, selectores del material interválico, de las figuras rítmicas, el tempo, etc.

Definición de métodos. Para desarrollar la aplicación se ha escogido el *modelo de desarrollo incremental*; es decir, el proceso es llevado a cabo en varios pasos, partiendo

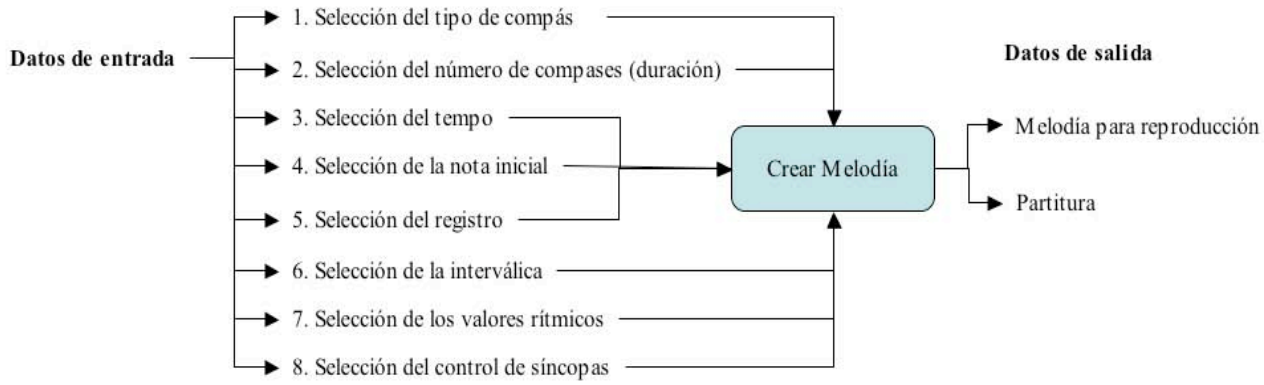
de la implementación de una versión simple del programa sin detalles, para continuar con la implementación de módulos subsecuentes de funciones avanzadas.

3.3.2. Diseño

3.3.2.1 Entrada y salida de datos en *ModusXXI*

En *ModusXXI* la entrada de datos es proporcionada por el usuario a través de la definición y selección de parámetros. Con los datos proporcionados por el usuario *ModusXXI* genera una melodía. Los datos de salida son la melodía para su reproducción y un archivo gráfico que muestra la melodía en notación de partitura.

Diagrama de entrada y salida de datos en *ModusXXI*



3.3.2.2 Programación Orientada a objetos

Una opción de programación práctica y útil para desarrollar *ModusXXI*, y que cumple con las funciones de entrada y salida señaladas en el diagrama anterior, es el esquema POO (Programación Orientada Objetos). En el libro *Java for Students*[15], Douglas Bell establece tres ideas centrales de la POO:

- *Encapsulamiento*. Cierta información relacionada es agrupada en un conjunto, protegiéndola del exterior. Dicha agrupación se convierte en un *objeto*. Un *objeto* es un conjunto de datos y métodos íntimamente relacionados. Los datos dentro del

[15] Bell Doublas, *Java for Students*, Essex, Pearson Education Limited, 1999 (3ra. Ed., *Java para estudiantes*, tr. española de Alfonso Vidal Romero; México, Pearson Educación de México, 2003), páginas 383-384.

objeto no pueden utilizarse directamente desde el exterior; son los *métodos* del objeto los que realizan las operaciones sobre los datos. Un *método* es una acción asociada a un *objeto*.

- *Herencia*. Es la incorporación de las características de una *clase* existente a una nueva *clase* que se desea crear. Una *clase* es la unidad de programación de un lenguaje orientado a *objetos*. Representa un conjunto de *objetos* similares (o idénticos). Describe los datos (*variables*) y *métodos* que contiene un *objeto*.
- *Polimorfismo*. Es un mismo *método* que se utiliza en distintos *objetos*.

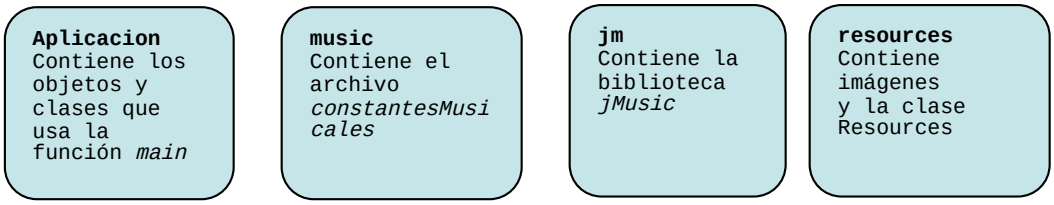
Para la programación de *ModusXXI* hemos escogido el lenguaje Java y el entorno de programación *NetBeans* versión 5. Java está desarrollado por Sun Microsystems y funciona dentro del esquema POO (Programación Orientada Objetos). Java resulta una buena opción para el desarrollo de *ModusXXI* pues ofrece las siguientes ventajas:[16]

- *Es un lenguaje de alto nivel*. La funciones de bajo nivel están ocultas para el programador, lo cual lo hace menos complejo y más fácil de aprender.
- *Orientación a objetos*. Esto permite desarrollar códigos relativamente cortos y reutilizables.
- *Es compatible con Internet*. Pueden crearse programas que funcionen en la red (Applets).
- *Es de propósito general*. Prácticamente cualquier aplicación puede ser programada en Java.
- *Es independiente de la plataforma*. Cualquier sistema operativo que cuente con la máquina virtual de Java puede ejecutar programas escritos en este lenguaje.
- *Es robusto*. Al ejecutarse los programas dentro de la máquina virtual de Java los efectos de cualquier error están confinados y controlados.
- *Cuenta con bibliotecas*. La mayor parte de su funcionalidad la proporcionan piezas de programa que están guardados en bibliotecas.

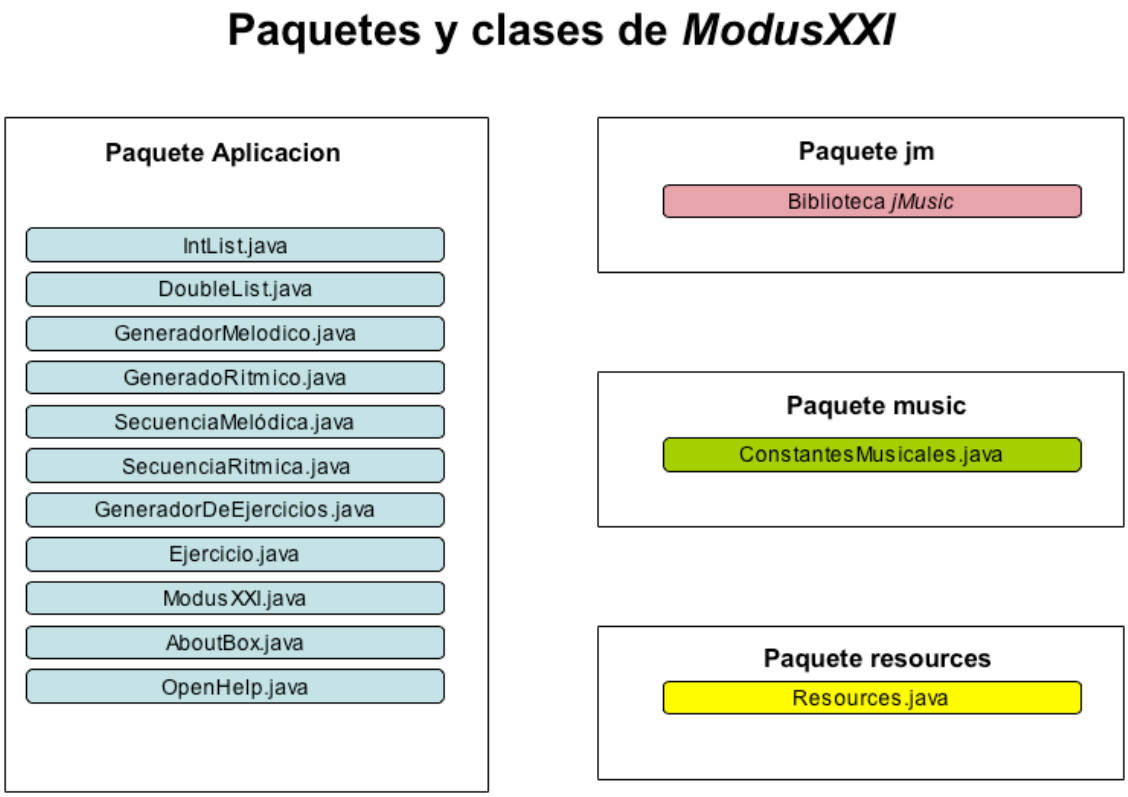
[16] Ibidem pag xxii.

3.3.2.3 Esquemas y diagramas de flujo de la aplicación

ModusXXI posee un esquema de programación orientado a objetos. Tiene una estructura de paquetes de archivos y clases (archivos **`.java`**). Los paquetes de archivos que conforman *ModusXXI* son: **`aplicacion`**, **`music`**, **`jm`** (biblioteca *jMusic)* y **`resources`**.

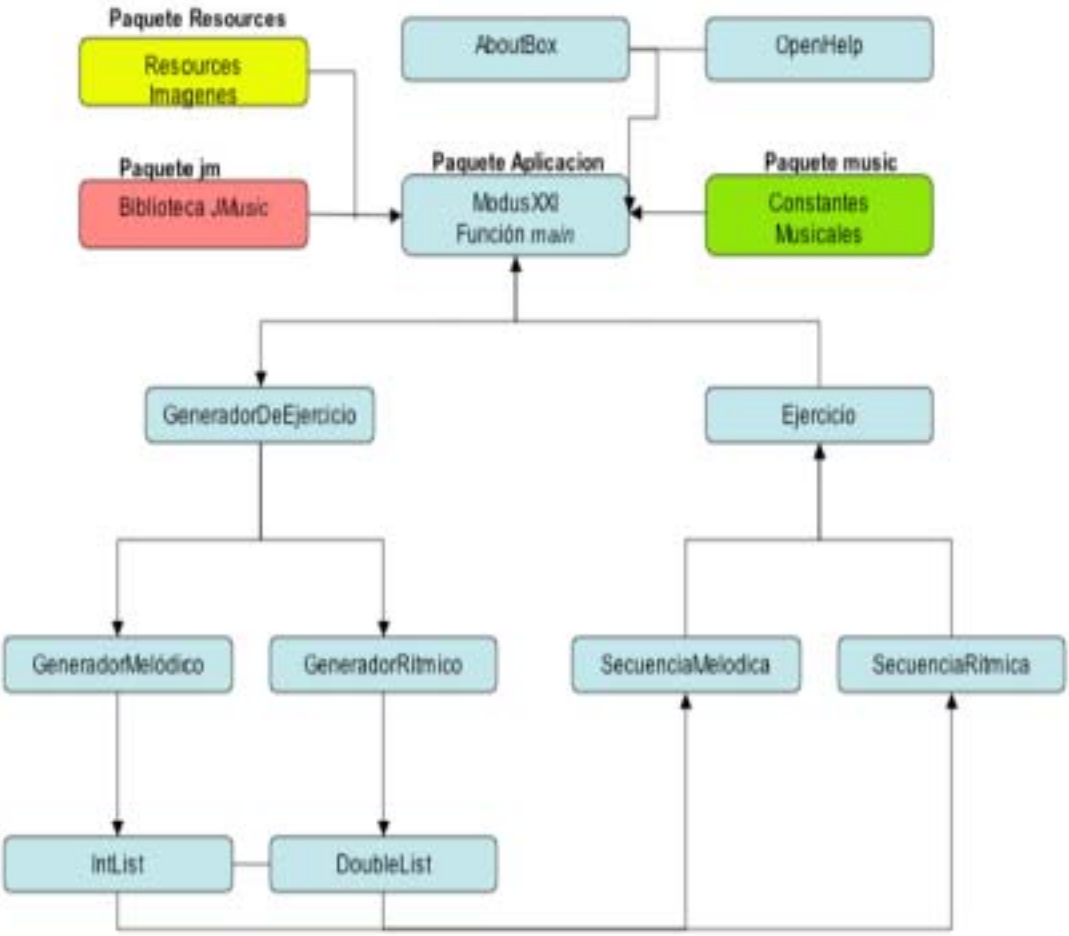


Cada uno de estos paquetes va a contar con un conjunto de clases, como lo muestra el siguiente esquema:



La interacción entre las clases y los paquetes se lleva a cabo de la siguiente manera:

Diagrama de flujo de clases en *ModusXXI*



3.3.3 Descripción de la implementación

A continuación presentamos una descripción concisa de los paquetes y clases que componen la aplicación. Si se desea consultar cuestiones específicas de la implementación es recomendable remitirse al código fuente que se encuentra en los apéndices de esta tesis.

3.3.3.1 Paquete `jm`: La biblioteca *jMusic*

El programa *ModusXXI* usa la biblioteca *jMusic*[17]. La biblioteca *jMusic* es un proyecto de investigación musical de la Universidad Tecnológica de Queensland, Australia, iniciado por Andrew Sorensen y Andrew Brown en 1998. Señalan sus autores:

> jMusic es un proyecto diseñado para proveer una biblioteca con un conjunto de herramientas de composición y procesamiento de audio a los compositores y desarrolladores de software. Provee un marco de trabajo sólido para la composición asistida en Java y puede también ser usado para música generativa, construcción de instrumentos [virtuales], ejecuciones interactivas y análisis musical. jMusic apoya a los músicos proporcionando una estructura de datos basada en eventos de notas/sonidos; y provee métodos para organizar, manipular y analizar la información musical. Las partituras [archivos] producidos por jMusic pueden ser convertidos a archivos MIDI o de audio para su almacenamiento, o posterior procesamiento o reproducción en tiempo real. jMusic puede leer o escribir archivos MIDI, archivos de audio, archivos XML y sus archivos propios .jm. Asimismo jMusic ofrece soporte a JavaSound, QuickTime y MidiShare. jMusic está diseñado para ser ampliado, motivándole a construir sobre su funcionalidad en Java, a crear composiciones musicales propias, herramientas e instrumentos. jMusic es 100% Java y funciona en Windows, MacOS, Linux, BSD, Solaris y cualquier otra plataforma que soporte Java.[18]

jMusic es un paquete de código abierto distribuido bajo la licencia GNU (General Public Licence). *ModusXXI* utiliza la biblioteca *jMusic* en la clase **`Ejercicio.java`** para:

- Definir valores rítmicos y notas.
- Establecer el compás.
- Establecer el tempo.
- Generar los archivos gráficos de notación musical.
- Generar el audio así como las funciones de reproducción y parada.

---

[17] http://jmusic.ci.qut.edu.au/ (Traducción del autor)

[18] http://jmusic.ci.qut.edu.au/

- Generar acentos en cada primer tiempo de compás.

3.3.3.2 Paquete `music`

El paquete `music` contiene el archivo `ConstanteMusicales.java`. `ConstanteMusicales.java` define los valores numéricos de los intervalos. Por ejemplo: **`segundaMenor = 1, segundaMayor = 2, terceraMenor = 3`**, etc. Se definen en este archivo los intervalos hasta la **`dobleOctava = 36.`**

3.3.3.3 Paquete `resources`

El paquete `resources` contiene los archivos gráficos que requiere el programa. Asimismo contiene la clase **`Resources`**, la cual sirve para enlazar el logo del programa con la opción "Acerca de *ModusXXI*" del menú "Ayuda".

3.3.3.4 Paquete `Aplicacion`

El paquete **`Aplicacion`** contiene los archivos esenciales de *ModusXXI* y está integrado por los siguientes clases (archivos **`.java`**):

Clases del paquete `Aplicación`
`IntList.java`
`DoubleList.java`
`GeneradorMelodico.java`
`GeneradorRitmico.java`
`SecuenciaMelodica.java`
`SecuenciaRitmica.java`
`GeneradorDeEjercicios.java`
`Ejercicio.java`
`ModusXXI.java`
`AboutBox.java`
`OpenHelp.java`

3.3.3.4.1 `IntList.java`[19]

Genera un arreglo de tamaño variable de números del tipo **`int`** (enteros). Es instanciado en la clase **`ModusXXI.java`** y sirve para almacenar la secuencia de notas.

3.3.3.4.2 `DoubleList.java`[20]

Genera un arreglo de tamaño variable de números del tipo **`double`** (punto flotante). Es instanciado en la clase **`ModusXXI.java`** y sirve para almacenar la secuencia de valores rítmicos.

Comparación entre valores del tipo int y double

Tipo	Tamaño	Valor Mínimo	Valor Máximo	Precisión
`Int`	32 bits	-2147483648	2147483647	Números enteros
`Double`	64 bits	±1.79E+308	±4.94E-324	14-15 cifras significativas

3.3.3.4.3 `GeneradorMelodico.java`

Esta clase crea de manera aleatoria las notas (alturas) de la melodía, a partir de la selección de la interválica, nota inicial y registro en el programa principal.

Crea una secuencia aleatoria de índices a partir de la lista **`valoresInterválicos`**; generada en **`ModusXXI.java`**, y que contiene los intervalos seleccionados por el usuario. Devuelve el arreglo de enteros **`ArrayNotas`**; el cual almacena las notas generadas azarosamente. Tiene dos métodos:

- **`generarNotas.`**
- **`generarNotasConRegistro`**.

[19] Esta es una clase tomada del libro *Java Examples in a Nutshell* de Flanagan David.

[20] Esta es una clase derivada de IntList.java.

A continuación se presenta y explica el código de esta clase implementado en lenguaje Java.

```
public class GeneradorMelodico {

    private static int [] valoresIntervalicos;

    public GeneradorMelodico(int [] intervalos) {
        valoresIntervalicos = intervalos;
    }

    public static int [] generarNotas(int notaInicial, int n){
      return generarNotasConRegistro(notaInicial, 0, 127, n);
    }

    // Aquí se encuentra el método "generarNotasConRegistro", explicado más
    adelante.
}
```

El código inicia con la declaración del arreglo **`private static int [] valoresIntervalicos`**. Este es un arreglo de números enteros, el cual contiene los intervalos seleccionados por el usuario, dentro de la clase **`ModusXXI.java`** (clase principal).

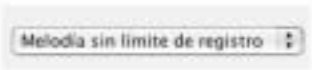
Este es el constructor de la clase:

```
public GeneradorMelodico(int [] intervalos) {
        valoresIntervalicos = intervalos;
    }
```

Crea una instancia de la clase **`GeneradorMelodico`**. Posee una variable que es el arreglo **`int [] intervalos`**. Dentro del constructor, el arreglo **`intervalos`** va a ser inicializado con el arreglo **`valoresIntervalicos`** comentado arriba.

Método `generarNotas`

`generarNotas` es el método utilizado cuando el usuario selecciona la opción



del programa principal. Crea un arreglo de números enteros, que contiene las notas de la melodía a generar. Posee dos variables de enteros: **`notaInicial`** y **`n`** que es el número de notas a generar (**`n`** va a estar dado por el tamaño del arreglo de figuras rítmicas creadas por la clase **`GeneradorRitmico`**). Utiliza el método **`generarNotasConRegistro`**, inicializando la variable **`notaMasGrave`** como 0 (valor de la nota MIDI más grave equivalente a Do0) y la variable **`notaMasAguda`** como 127 (valor de

la nota MIDI más aguda equivalente a Sol10). Este método sirve para la generación de "melodías sin límite de registro" (o en términos más exactos, el límite de registro es definido por los límites del protocolo MIDI).

Método `generarNotasConRegistro`

`generarNotasConRegistro` es el método usado cuando está seleccionada la opción

del programa principal. Crea un lista (arreglo) de notas, mediante un proceso de selección aleatoria de índices del arreglo **`valoresIntervalicos`**; es decir, los intervalos son escogidos azarosamente, por el método a partir de una lista de intervalos (**`valoresIntervalicos`**) definida por el usuario en el programa principal. Observemos el código de este método:

```
public static int [] generarNotasConRegistro(int notaInicial,
            int notaMasGrave, int notaMasAguda, int numeroDeNotas){

        int arrayNotas [] = new int [numeroDeNotas];

        arrayNotas [0] = notaInicial;
        for ( int i = 1 ; i < numeroDeNotas ; i++) {

            int iteraciones = 0;
            int indiceAleatorio;
            int nuevaNota;

            do  {

                if (iteraciones++ > 1000) {
                    return null;
                }

                indiceAleatorio =
                (int)Math.floor(Math.random()*valoresIntervalicos.length);
                nuevaNota = arrayNotas[i - 1]  + valoresIntervalicos
                [indiceAleatorio];
             } while (nuevaNota > notaMasAguda || nuevaNota < notaMasGrave );

            arrayNotas [i] = nuevaNota;
        }
        return arrayNotas;
    }
```

El método **`public static int [] generarNotasConRegistro`** posee cuatro variables: **`notaInicial, notaMasGrave, notaMasAguda, numeroDeNotas`**. Se declara el arreglo de enteros **`int arrayNotas [] = new int [numeroDeNotas]`** y se inicializa el primer índice de dicho arreglo con la **`notaInicial`** de la siguiente forma: **`arrayNotas [0]`**

= notaInicial. Se crea un ciclo **for**, donde el contador **int i = 1** define los índices del arreglo **arrayNotas**:

for (int i = 1 ; i < numeroDeNotas ; i++) "Para el índice **i = 1** se agregará un nuevo índice al **arrayNotas** siempre y cuando **i <** (sea menor al) **numeroDeNotas**". Posteriormente se declaran tres variables locales nuevas: **iteraciones, indiceAleatorio** y **nuevaNota** y se inicia un ciclo anidado **do-while**:

```
do  {
      if (iteraciones++ > 1000) {
            return null;
      }
```

"Si el contador **iteraciones** (inicializado con el valor 0) llega a un número de iteraciones mayor que mil[21], se regresará un valor **null**, es decir un objeto inexistente; lo cual significará que no se ha encontrado una **nuevaNota** y por tanto no es posible generar el arreglo **arrayNotas**." Si este no es el caso, entonces:

```
indiceAleatorio=(int)Math.floor(Math.random()*valoresIntervalicos.length);
```

Se generará la variable entera **indiceAleatorio** utilizando las funciones **Math.random()** y **Math.floor()** de la biblioteca **java.lang.Math**. El método **Math.random()** devuelve un número pseudoaleatório del tipo **double** con signo positivo mayor o igual que 0.0 y menor que 1.0; con una distribución aproximadamente uniforme.[22] El método **Math.floor()** devuelve el valor del tipo **double**, que es menor ó igual a un entero matemático.[23] Es decir, convierte el número de punto flotante devuelto por el método **Math.random()** en un entero. De acuerdo al tamaño ó largo del arreglo **valoresIntervalicos.length**, la variable **indiceAleatorio** seleccionará aleatoriamente algún índice del arreglo **valoresIntervalicos**. Así:

```
nuevaNota = arrayNotas[i - 1] + valoresIntervalicos [indiceAleatorio];
```

[21] Este número ha sido elegido arbitrariamente, bajo la consideración de que mil iteraciones son suficientes para determinar la existencia ó inexistencia de valores.

[22] API Java. JavaDocs/Java5_docs/api/index.html

[23] API Java. JavaDocs/Java5_docs/api/index.html

Se generará una nueva nota concatenando el índice menos uno **`[i - 1]`** (esto porque ya se cuenta con el primer índice que es la **`notaInicial`**) del arreglo **`arrayNotas`** con el índice del arreglo **`valoresIntervalicos`**, definido ahora por el **`indiceAleatorio`**.

A continuación tenemos la condición **`while`** de este ciclo **`do`**:

```
while (nuevaNota > notaMasAguda || nuevaNota < notaMasGrave ) {
            arrayNotas [i] = nuevaNota;
        }
```

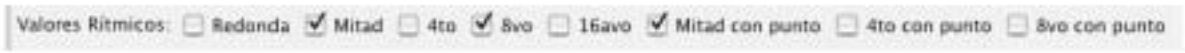
Mientras que la **`nuevaNota`** sea mayor que la **`notaMasAguda`** o menor que la nota **`notaMasGrave`** se agregará una **`nuevaNota`** al arreglo **`arrayNotas`**.

```
return arrayNotas;
```

Finalmente el método devuelve el arreglo **`arrayNotas`**.

3.3.3.4.4 `GeneradorRitmico.java`

Esta clase genera de manera aleatoria las figuras rítmicas de la melodía; a partir del conjunto de figuras rítmicas escogidas por el usuario en la clase principal:



Genera una secuencia aleatoria de índices a partir de la lista **`valoresRitmicos`**, (creado en la clase principal **`ModusXXI.java`**). Devuelve un arreglo con las figuras rítmicas escogidas en números del tipo **`double`** (Esto porque los valores rítmicos en la biblioteca *jMusic* son del tipo **`double`**). Posee dos métodos:

- **`generarRitmosPorCompas`**. Crea un arreglo de figuras rítmicas. Posee la variable **`duracionTotal`**. Los ritmos producidos por este método pueden ser sincopados.
- **`generarRitmosXCompasNoSincopado`**. Además de la variable **`duracionTotal`** cuenta con la variable **`compas`**.

A continuación se presenta y explica el código de esta clase implementado en lenguaje Java.

```
public class GeneradorRitmico {

    private static double [] valoresRitmicos;

    public GeneradorRitmico(double [] valores) {
        valoresRitmicos = valores;
    }

    // Aquí se encuentra el método “generarRitmosPorCompas”,
    explicado más adelante.

    // Aquí se encuentra el método “generarRitmosXCompasNoSincopado”,
    explicado más adelante.

}
```

El código inicia con la declaración del arreglo **`private static double [] valoresRitmicos`**. Este es un arreglo de números del tipo **`double`**, el cual contiene las figuras rítmicas seleccionados por el usuario.

```
    public GeneradorRitmico(double [] valores) {
        valoresRitmicos = valores;
    }
```

Este es el constructor de la clase y crea de una instancia de la clase **`GeneradorRitmico`**. Posee una variable que es el arreglo **`double [] valores`**. Dentro del constructor, el arreglo **`valores`** va a ser inicializado con el arreglo **`valoresRitmicos`**.

Método `generarRitmosPorCompas`

`generarRitmosPorCompas` es el método utilizado cuando el usuario selecciona la opción 

del programa principal. Teniendo como límite la **`duracionTotal`** de la melodía (multiplicando el número de compases por numerador del compás), genera una secuencia de valores rítmicos con posibles síncopas entre compases; a partir de un proceso de selección aleatoria de índices de la lista **`valoresRitmicos`**. Veamos el método completo:

```
public static double [] generarRitmosPorCompas(double duracionTotal){
        int valorAleatorio = 0;
        double duracion = 0;

        DoubleList temp = new DoubleList();
        for (int i = 0 ; duracion < duracionTotal ; i++) {
```

```
        valorAleatorio = (int) Math.floor(Math.random() *
              valoresRitmicos.length);
        temp.add(valoresRitmicos[valorAleatorio]);
        duracion = duracion + valoresRitmicos [valorAleatorio];
    }

    return temp.toArray();
      }
```

`public static double [] generarRitmosPorCompas` posee la variable **`duracionTotal`** la cual es un número del tipo **`double`**. Se declaran e inicializan dos variables internas numéricas inicializadas en cero: **`valorAleatorio`** del tipo **`int`** y **`duracion`** del tipo **`double`**. Se declara e inicializa el objeto **`temp`** que es una instancia de la clase **`DoubleList`** y que crea un arreglo de tamaño variable de números del tipo **`double`** con figuras rítmicas. Se crea un ciclo **`for`**, donde el contador **`int i = 0`** define los índices del arreglo **`duracionTotal`**:

`for (int i = 0 ; duracion < duracionTotal ; i++)` "Para el índice **`i = 0`** se agregará un nuevo índice al **`arrayNotas`** siempre y cuando **`i <`** (sea menor a la) **`duracionTotal`** de la melodía," cumpliendo el siguiente proceso:

`valorAleatorio = (int)Math.floor(Math.random()*valoresRitmicos.length).` Se generará la variable entera **`valorAleatorio`** utilizando las funciones **`Math.random()`** y **`Math.floor()`** de la biblioteca **`java.lang.Math`**. El método **`Math.random()`** devuelve un número pseudoaleatório del tipo **`double`** con signo positivo mayor o igual que 0.0 y menor que 1.0; con una distribución aproximadamente uniforme.[24] El método **`Math.floor()`** devuelve el valor del tipo **`double`**, que es menor o igual a un entero matemático.[25] Es decir, convierte el número de punto flotante devuelto por el método **`Math.random()`** en un entero. De acuerdo al tamaño o largo del arreglo **`valoresRitmicos.length`**, la variable **`valorAleatorio`** seleccionará aleatóriamente algún índice del arreglo **`valoresRitmicos`**. **`temp.add(valoresRitmicos[valorAleatorio]).`** Al arreglo **`temp`** se agregará un valor rítmico de acuerdo al índice escogido por el **`valorAleatorio`**.

[24] API Java. JavaDocs/Java5_docs/api/index.html

[25] API Java. JavaDocs/Java5_docs/api/index.html

`duracion = duracion + valoresRitmicos [valorAleatorio]`. La duración de la melodía es determinada por la suma de ésta más el índice del arreglo **`valoresRitmicos`**, proporcionado por **`valorAleatorio`**. La **`duracion`** tiene como limite a la variable **`duracionTotal`**.

`return temp.toArray()`. Finalmente el método devuelve el arreglo **`temp`** (usando la función **`toArray()`** del la clase **`DoubleList)`** con los valores rítmicos generados.

Método `generarRitmosXCompasNoSincopado`

`generarRitmosXCompasNoSincopado` es el método utilizado cuando el usuario selecciona la opción 

del programa principal. Funciona de manera similar a **`generarRitmosPorCompas`**, pero requiere la variable global **`compas`** y las variables locales **`nuevaFiguraRitmica`** y **`pulso`**. Crea una secuencia de valores rítmicos sin síncopas entre compases (es decir en compases téticos); a partir de un proceso de selección aleatoria de índices de la lista **`valoresRitmicos`**; considerando el tamaño del compás, o lo que reste del mismo. Si es seleccionada una figura rítmica mayor que el compás o su residuo, el método buscará una nueva figura rítmica que sea menor o igual que dicho compás o residuo; generando consecuentemente compases téticos. Veamos el método completo:

```
public static double [] generarRitmosXCompasNoSincopado(double duracionTotal,
            double compas){

        int valorAleatorio = 0;
        double duracion = 0;
        double nuevaFiguraRitmica = 0;
        double pulso = 0;

        DoubleList temp = new DoubleList();
        for (int i = 0 ; duracion < duracionTotal ; i++) {

            int iteraciones = 0;

            do {

                if (iteraciones++ > 1000) {
                    return null;
                }

                valorAleatorio = (int) Math.floor(Math.random() *
                    valoresRitmicos.length);
                nuevaFiguraRitmica = valoresRitmicos[valorAleatorio];
            } while ( pulso + nuevaFiguraRitmica > compas );
```

```
            temp.add(nuevaFiguraRitmica);
            duracion = duracion + nuevaFiguraRitmica;
            pulso = pulso + nuevaFiguraRitmica;
            if ( pulso >= compas )
                pulso = pulso - compas;
        }
        return temp.toArray();
    }
```

`public static double [] generarRitmosXCompasNoSincopado` posee dos variables numéricas del tipo **`double`**: **`duracionTotal`** y **`compas`**. Se declaran tres variables locales del tipo **`double`**: **`duracion`**, **`nuevaFiguraRitmica`** y **`pulso`**; y la variable local del tipo **`int valorAleatorio`**. Todas se incializan con el valor numérico cero. Como en el método **`generarRitmosPorCompas`** se crea un objeto **`temp`**, instancia de la clase **`DoubleList`** y se crea de la misma forma el ciclo **`for`**:

`for (int i = 0 ; duracion < duracionTotal ; i++)` No obstante, se anida in ciclo **`do-while`** de la siguiente manera:

Se declaran la variable local **`int iteraciones = 0`** y se inicia el ciclo **`do-while`**:

```
do  {
      if (iteraciones++ > 1000) {
            return null;
      }
```

Si el contador **`iteraciones`** (inicializado con el valor cero) llega a un número de iteraciones mayor que mil[26], se regresará un valor **`null`**, es decir un objeto inexistente; lo cual significará que no se ha encontrado una **`nuevaFiguraRitmica`** y por tanto no es posible generar el arreglo **`temp`**. Si este no es el caso, entonces:

`valorAleatorio = (int) Math.floor(Math.random() * valoresRitmicos.length)`. Se generará la variable entera **`indiceAleatorio`** utilizando las funciones **`Math.random()`** y **`Math.floor()`** de la biblioteca **`java.lang.Math`**; de la misma forma que en el método anterior.

[26] Este número ha sido elegido arbitrariamente, bajo la consideración de que mil iteraciones son suficientes para determinar la existencia ó inexistencia de valores.

`nuevaFiguraRitmica = valoresRitmicos[valorAleatorio]`. Se generará la variable **`nuevaFiguraRitmica`** la cual corresponderá al índice aleatorio del arreglo **`valoresRitmicos`**. Esta **`nuevaFiguraRitmica`** será generada siempre y cuando (**`while`**) **`pulso + nuevaFiguraRitmica >`** (sea mayor que) **`compas`**. Es decir, si la nueva figura rítmica seleccionada aleatoriamente es mayor que el compás, el método la desecha y busca una **`nuevaFiguraRitmica`** que cumpla esta condición; lográndose entonces que todos los compases sean téticos.

`temp.add(nuevaFiguraRitmica)`. Se agrega la **`nuevaFiguraRitmica`** al arreglo **`temp`**.

`duracion = duracion + nuevaFiguraRitmica`. La duración de la melodía es determinada por la suma de ésta más la **`nuevaFiguraRitmica`**. Tiene como limite, establecido en la condicion if, la **`duracionTotal`**. La variable **`pulso = pulso + nuevaFiguraRitmica`**; funciona exactamente igual que la variable **`duracion`**, pero define la unidad o nivel de subdivisión. Se le aplica el siguiente operador condicional: **`if ( pulso >= compas )`** "Si el **`pulso`** es mayor o igual que el **`compas`**, entonces el **`pulso = pulso – compas`**.

El método devuelve el arreglo **`temp`** (usando la función **`toArray()`** del la clase **`DoubleList)`** con los valores rítmicos generados.

Cabe mencionar, que desde el punto de vista de ejecución de acciones en el programa, primero es generado el arreglo de figuras rítmicas y después el arreglo de notas, pues el número de figures rítmicas va a definir el número de notas a generar.

3.3.3.4.5 `SecuenciaMelodica.java`

Llama a la clase **`GeneradorMelodico.java`** para crear una instancia u objeto de **`SecuenciaMelodica`**.

3.3.3.4.6 `SecuenciaRitmica.java`

Llama a la clase **`GeneradorRitmico.java`** para crear una instancia u objeto de **`SecuenciaRitmica`**.

3.3.3.4.7 `Ejercicio.java`

Crea el objeto **`Ejercicio`** concatenando los arreglos **`secuenciaRitmica`** y **`secuenciaMelodica`**. Traduce los valores obtenidos en **`GeneradorMelodico`** y **`GeneradorRitmico`** a los parámetros de la biblioteca *jMusic*, implementando las funciones de reproducción y parada, así como la generación del archivo gráfico de partitura. Esta es la estructura general de la clase:

```
public class Ejercicio implements JMC, ConstantesMusicales {
	private static double [] secuenciaRitmica;
	private static int [] secuenciaMelodica;
	private static int numerador = 4;
	private static int denominador = 4;
	public static double pulso;

	public Ejercicio(double [] ritmos, int [] notas) {
	  secuenciaRitmica = ritmos;
	  secuenciaMelodica = notas;
	}
	// Aquí se encuentra los métodos lenght, ritmo, notas, ritmo, notas,
	asignarNumerado y asignarDenominador; explicados más adelante

	public void playJM(int tempo) {
		// Este método es explicado más adelante
	}
	public void stopJM(int tempo) {
		// Este método es explicado más adelante
	}
	public void viewJM(int tempo) {
		// Este método es explicado más adelante
	}
}
```

Las variables locales **secuenciaRitmica** y **secuenciaMelodica** van a contener los arreglos **ritmos** y **notas**, respectivamente; creados por **GeneradorRitmico** y **GeneradorMelodico** en la clase **GeneradorDeEjercicios.java** (explicada más adelante).

Constructor:

```
public Ejercicio(double [] ritmos, int [] notas) {
      secuenciaRitmica = ritmos;
      secuenciaMelodica = notas;
   }
```

La clase **Ejercicio** posee los siguientes métodos:

- **public static int length() {**
  return secuenciaRitmica.length;
  } Devuelve el tamaño del arreglo **secuenciaRitmica**

- **public static double ritmo(int n) {**
  return secuenciaRitmica[n];
  } Devuelve el índice del objeto **secuenciaRitmica**

- **public static int nota(int n) {**
  return secuenciaMelodica[n];
  } Devuelve el índice del objeto **secuenciaMelodica**

- **public static double [] ritmos() {**
  return secuenciaRitmica;
  } Devuelve el arreglo **secuenciaRitmica**

- **public static int [] notas() {**
  return secuenciaMelodica;
  } Devuelve el arreglo **secuenciaMelodica**

- **public static void asignarNumerador(int num) {**
  numerador = num;
  } Asigna el **Numerador**

- **public static void asignarDenominador(int den) {**

```
        denominador = den;
```
`}` Asigna el **`Denominador`**

Método `playJM()`

Reproduce el **`ejercicio`**. Traduce los parámetros de ejercicio a los parámetros de la biblioteca *jMusic*. Veamos el código completo de este método:

```
public void playJM(int tempo) {
       if (secuenciaRitmica == null || secuenciaMelodica == null) {
            return;
        }
        Note [] jm_notas = new Note[length()];
        for ( int i = 0 ; i < length() ; i++){
            jm_notas[i] = new Note(nota(i), ritmo(i));
        }
        double [] acentos = {0.0}; //para denominador = cuarto
        double numeroDeCuartosPorCompas = 4.0 * (double) numerador /
          denominador;
         Phrase jm_frase = new Phrase(jm_notas);
        Part jm_parte = new Part(jm_frase);
        Score jm_score = new Score(jm_parte);
        Mod.accents(jm_score, numeroDeCuartosPorCompas,
        jm_score.setTimeSignature(numerador, denominador);
        jm_score.setVolume(50);
        jm_score.setTitle("Modus XXI");
        jm_score.setTempo(tempo);
        Play.midi(jm_score, false);
}
```

`public void playJM(int tempo)` posee una variable **`tempo`** que es definida en la clase principal. Se inicializa una condición **`if`** :

```
        if (secuenciaRitmica == null || secuenciaMelodica == null) {
             return;
         }
```

"Si la **`secuenciaRitmica`** o la **`secuenciaMelodica`** son iguales a **`null`** (objetos vacíos), entonces el programa se detiene". Si este no es el caso entonces:

`Note [] jm_notas = new Note[length()];`

Se crea el arreglo **`jm_notas`** del objeto tipo **`Note`** de la biblioteca *jMusic*. Usando la función **`length()`**, creada más arriba.

Se inicia un ciclo **for**:

```
for ( int i = 0 ; i < length() ; i++){
            jm_notas[i] = new Note(nota(i), ritmo(i));
        }
```

"Dado un índice **i** inicializado en cero, se generarán **jm_notas**, siempre y cuando el índice **i** sea menor que el largo de la **secuenciaRitmica**".

Se declaran y definen las siguientes variables para usarse en el formato de la biblioteca *jMusic*:

```
double [] acentos = {0.0}; //para denominador = cuarto
double numeroDeCuartosPorCompas = 4.0 * (double) numerador /
denominador;
```

Se declaran y crean los siguientes objetos de la biblioteca *jMusic*:

```
Phrase jm_frase = new Phrase(jm_notas);
Part jm_parte = new Part(jm_frase);
Score jm_score = new Score(jm_parte);
```

Phrase es un objeto que agrupa a las **jm_notas**. **Phrase** a su vez es encapsulado en el objeto **Part** y éste en el objeto **Score**, que es el objeto de mayor nivel en la biblioteca *jMusic*, **Score** cuenta con los siguientes métodos:

Mod.accents(jm_score, numeroDeCuartosPorCompas, acentos, Note.MAX_DYNAMIC). Este método crea acentos en el primer tiempo de cada compás.

jm_score.setTimeSignature(numerador, denominador). Define el compás en el archivo gráfico de partitura.

jm_score.setVolume(50). Define el volumen en valores MIDI.

jm_score.setTitle("Modus XXI"). Inserta el título *ModusXXI* en la ventana que contiene la partitura.

jm_score.setTempo(tempo). Define el tempo.

Play.midi(jm_score, false). Reproduce la melodía.

Método `stopJM()`

Detiene la reproducción del **`ejercicio`**. Utiliza la función **`Play.stopMidi().`**

```
public void stopJM() {
      Play.stopMidi();
}
```

Método `viewJM()`

Genera el archivo gráfico de partitura con el **`ejercicio`**. Utiliza la función **`Notate().`**

```
public void viewJM() {
      Notate(jm_score 100, 100);
}
```

3.3.3.4.8 `GeneradorDeEjercicios.java`

Instancia las clases **`GeneradorRitmico`** y **`GeneradorMelodico`** para la generación de un objeto **`ejercicio`**. Veamos el esquema general del código:

```
public class GeneradorDeEjercicios implements JMC {
    private static GeneradorRitmico gr;
    private static GeneradorMelodico gm;

    public GeneradorDeEjercicios(double [] valoresRitmicos, int [] notas) {
        gr = new GeneradorRitmico (valoresRitmicos);
        gm = new GeneradorMelodico (notas);
    }

    public static Ejercicio generarEjercicio (
      // Método sobrecargado cuatro veces, explicado más adelante
    }
}
```

Son declarados los objetos **`gr`** y **`gm`**, y se presenta el constructor de la clase:

```
public GeneradorDeEjercicios(double [] valoresRitmicos, int [] notas) {
        gr = new GeneradorRitmico (valoresRitmicos);
        gm = new GeneradorMelodico (notas);
    }
```

Los objetos **gr** y **gm** son instancias respectivas de **GeneradorRitmico** y de **GeneradorMelodico**. A cada uno de estos se les asignan los arreglos **valoresRitmicos** y **notas**, creados en la clase **ModusXXI.java**.

Método `generarEjercicio`

El método **generarEjercicio** está sobrecargado cuatro veces (es decir, hay cuatro métodos con el mismo nombre) y genera un objeto **Ejercicio**. Dependiendo de los parámetros introducidos por el usuario, es utilizado alguno de los siguientes métodos:

- 1ª Sobrecarga: genera un ejercicio con registro y evitando síncopas. Tiene las siguientes variables: **numerador, denominador, numeroDeCompases, pulso, notaInicial, notaMasGrave** y **notaMasAguda**.
- 2da Sobrecarga: genera un ejercicio sin registro y evitando síncopas. Carece de las variables **notaInicial, notaMasGrave** y **notaMasAguda**.
- 3ra Sobrecarga: genera un ejercicio con registro y permitiendo síncopas. Carece de la variable **pulso**.
- 4ta Sobrecarga: genera un ejercicio sin registro y permitiendo síncopas. Carece de las variables **pulso, notaInicial, notaMasGrave** y **notaMasAguda**.

Veamos el código de la 1ª Sobrecarga:

```
public static Ejercicio generarEjercicio (int numerador, int denominador, int
numeroDeCompases, int notaInicial, int notaMasGrave, int notaMasAguda) {

   double compas = QUARTER_NOTE * 4.0 * (double) numerador / denominador
   double duracionTotal =  numeroDeCompases * compas;
   int [] notas;
   double [] ritmos = gr.generarRitmosPorCompas (duracionTotal);
   if (ritmos == null) {
          notas = null;
   } else {
          notas = gm.generarNotasConRegistro(notaInicial, notaMasGrave,
          notaMasAguda, ritmos.length);
   }

   Ejercicio ejercicio = new Ejercicio (ritmos, notas);
   ejercicio.asignarNumerador (numerador);
   ejercicio.asignarDenominador (denominador);
   return ejercicio;
}
```

En sus cuatro formas, el método **generarEjercicio** realiza las siguientes funciones:

- Define el **compas**:
  double compas = QUARTER_NOTE * 4.0 * (double) numerador / denominador;
- Define la **subdivision** (sólo en el 3er y 4to métodos):
  double subdivision = QUARTER_NOTE * 4.0 * pulso;
- Define la **duracionTotal**:
  double duracionTotal = numeroDeCompases * compas;
- Declara el arreglo **int [] notas;**
- Declara y define el arreglo **double [] ritmos**. Usa los métodos:
  - **gr.generarRitmosXCompasNoSincopado (duracionTotal, subdivision)** (1ra y 2da Sobrecargas). Llama al método **generarRitmosXCompasNoSincopado** del objeto **gr**, instancia de la clase **GeneradorRitmico**; usando las variables **duracionTotal** y **subdivision** arriba definidas.
  - **gr.generarRitmosPorCompas (duracionTotal)**. (3ra y 4ta Sobrecargas). Llama al método **generarRitmosPorCompas** del objeto **gr**, instancia de la clase **GeneradorRitmico**; usando la variable **duracionTotal**.

- Crea un ciclo **if** para generar el arreglo **notas**:
  if (ritmos == null) {
  notas = null;
  } else {
  notas = gm.generarNotas(notaInicial, ritmos.length); (2do y 4to métodos).
  ó
  notas = gm.generarNotasConRegistro(notaInicial, notaMasGrave, notaMasAguda, ritmos.length); (1ro y 3er métodos).

  "Si **ritmos** es igual a **null** (objeto vacío), entonces **notas** también será otro objeto vació y se detendrá el programa; de lo contrario el arreglo **notas** será creado, con el método **generarNotas** ó **generarNotasConRegistro** del objeto **gm**, instancia de la

clase **`GeneradorMelodico`**; usando las variables **`notaInicial`**, **`notaMasGrave`**, **`notaMasAguda`** y el largo del arreglo **`ritmos`**."

- Crea un **`ejercicio`** de la clase **`Ejercicio`** concatenando los arreglos **`ritmos`** y **`notas`**.
- Asigna el **`numerador`** y el **`denominador`** del compás al objeto **`ejercicio`**, usando los siguientes métodos de la clase **`Ejercicio`**:

```
ejercicio.asignarNumerador (numerador);
ejercicio.asignarDenominador (denominador);
```

- Finalmente devuelve un objeto **`ejercicio`**.

3.3.3.4.9 `ModusXXI.java`

Ésta es la clase principal del programa. Lleva a cabo las funciones de recopilación de los datos introducidos por el usuario, generación, presentación, reproducción y parada de la melodía; así como la construcción de la interfaz gráfica de usuario. (La implementación de la interfaz es descrita en el siguiente apartado). **`ModusXXI.java`** crea un objeto **`generadorDeEjercicios`** concatenando los arreglos **`valoresRitmicos`** y **`valoresIntervalicos`**. Crea un objeto **`ejercicio`**, el cual llama a la función **`generarEjercicio`**; asignándole las variables **`numerador`, `denominador`, `numeroDeCompases`, `pulso`, `notaInicial`, `notaMasGrave`, `notaMasAguda`**. Asimismo, llama a los métodos **`playJM()`**, **`stopJM()`** y **`viewJM()`**, para reproducir, detener o ver la partitura. Veamos la estructura general de la clase:

```
public class ModusXXI extends JFrame implements JMC, ActionListener, ConstantesMusicales {
	// Declaración de variables tanto de la interfaz como de la generación de la
	melodía, señaladas más adelante
	public ModusXXI() {
	super("ModusXXI");
		initComponents();
		setVisible(true);
	}
	public void initComponents() {
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
	public void actionPerformed(ActionEvent evt){
	}
	GeneradorDeEjercicios gen = new GeneradorDeEjercicios(valoresRitmicos,
			valoresIntervalicos);
	public static void main(String[] args ) {
		new ModusXXI();
	}
	public int verificarCrearMelodia() {
	}
	private void abrirAbout() {
		//Código omitodo por brevedad
	}
	private void abrirAyuda() {
		//Código omitodo por brevedad
	}
	private void abrirWebSite() {
		//Código omitodo por brevedad
	}
}
```

Antes de iniciar la clase, **`ModusXXI.java`** importa los paquetes *music* y *jMusic*.

```
import music.ConstantesMusicales;
import jm.JMC;
import jm.music.data.Score;
```

Dentro de la clase se declaran las siguientes variables globales (Datos de entrada por parte del usuario):

```
public static int numeroDeCompases;
public static int numerador;
public static int denominador;
public static int notaInicial;
public static int notaMasGrave;
public static int notaMasAguda;
public static double [] valoresRitmicos;
public static int [] valoresIntervalicos;
public static double pulso;
```

Se definen por default las siguientes variables (valores de inicio con los que cuenta el usuario):

```
numeroDeCompases = 4;
numerador = 4;
denominador = 4;
pulso = 1.0/4; //ajusta el control de síncopa
notaInicial = C4;
notaMasGrave = C4;
notaMasAguda = C5;
```

Se declaran los objetos:

```
public static Ejercicio ejercicio = null;
public Score jm_score;
```

Éste es el constructor de la clase:

```
public ModusXXI() {
        super("ModusXXI");
        initComponents();
        setVisible(true);
}
```

Contiene los métodos **`super()`** (define la superclase) **`initComponents()`** (inicializa los componentes de la interfaz) y **`setVisible(true)`** (muestra el programa).

ModusXXI.java contiene los siguientes métodos:

- **public void initComponents() {**
  this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  } Inicializa los componentes de la interfaz. El método presentado cierra la ventana al terminar el proceso.

- **public void actionPerformed(ActionEvent evt){**
  } Atrapa los eventos producidos por el usuario en la interfaz. Asimismo, realiza las funciones de reproducción, parada y mostrar partitura.

GeneradorDeEjercicios gen = new GeneradorDeEjercicios(valoresRitmicos, valoresIntervalicos). Crea el objeto **gen** , instancia de la clase **GeneradorDeEjercicios**, para la generación del objeto **ejercicio**.

valoresRitmicos es un arreglo de números del tipo **double** que contiene las figuras rítmicas seleccionadas por el usuario, coleccionadas dentro del arreglo **tempListR**, a través del método **toArray()** de la clase **DoubleList**. **valoresIntervalicos** es un arreglo de números enteros que contiene los intervalos seleccionadas por el usuario, coleccionados dentro del arreglo **tempList**, a través del método **toArray()** de la clase **IntList**.

- **public static void main(String[] args) {**
  new ModusXXI().
  } Función principal del programa. Crea una instancia de la clase **ModusXXI**.

- **public int verificarCrearMelodia() {**
  } Verifica si ha sido creada una melodía. Este método sirve para atrapar excepciones y evitar errores en el programa.

- **abrirAbout(), abrirAyuda()** y **abrirWebSite()** abren los vínculos “Acerca de ModusXXI”, “Ayuda” y “WebSite” del menú Ayuda.

3.3.4 La interfaz de usuario de *ModusXXI*

3.3.4.1 Navegación en *ModusXXI*

Desde el punto de vista de navegación, el usuario tiene diez opciones en *ModusXXI*. Las primeras ocho se refieren a la entrada de datos del usuario: *selección de compás, selección del número de compases* (duración), *selección del tempo, selección de la nota inicial, selección del control de registro tonal, selección de la interválica, selección de los valores rítmicos* y *selección del control de síncopas.* Las opciones *selección de registro tonal, selección de la interválica* y *selección del control de síncopas* ofrecen más posibilidades de navegación:

Selección de límite de registro

- Sin límite de registro
- Con límite de registro
  - Nota más grave
  - Nota más aguda

Selección de la interválica

- Selección manual
- Selección *Modus Novus*
- Selección “Sugerencias de estudio”

Selección de síncopa

- Permitir síncopa entre compases
- Evitar síncopa entre compases
  - Sin subdivisión
  - Subdivisión en cuartos
  - Subdivisión en octavos

Por su parte, la opción nueve que se refiere a la creación de la melodía ofrece las siguientes opciones de navegación:

Crear melodía

- Reproducir
- Detener reproducción

- Ver partitura
  - Funciones de la biblioteca *jMusic*

La opción 10, el menú de ayuda cuenta con las siguientes opciones de navegación: *Ayuda*

- Abrir ventana *Acerca de ModusXXI*
- Abrir archivo de ayuda
- Abrir vínculo a la página en Internet del proyecto (en construcción)

Obsérvese el siguiente diagrama de navegación:

3.3.4.2 La clase `ModusXXI.java`

La interfaz de usuario de *ModusXXI* está también desarrollada en el lenguaje de programación Java, diseñada sobre el modelo de interfaz objeto-acción OAI. *ModusXXI* utiliza las bibliotecas **`java.awt`** y **`javax.swing`** para la generación de la interfaz, la cual se encuentra en el archivo **`ModusXXI.java`**, dentro del paquete **`Aplicacion`**.

La biblioteca **`java.awt`** es utilizada para el diseño de ventana, para interceptar eventos y para generar el menú "Ayuda".

Para el diseño de ventanas *ModusXXI* utiliza dos clases de **`java.awt`**:

- **`java.awt.BorderLayout`**. Esta clase es utilizada para el diseño del panel central.
- **`java.awt.FlowLayout`**. Esta clase se usa en los paneles interiores.

Para interceptar eventos *ModusXXI* utiliza las siguientes clases de **`java.awt`**:

- **`java.awt.event.ActionEvent`**. Habilita los eventos producidos al presionar algún botón.
- **`java.awt.event.ActionListener`**. Identifica la ejecución de dichos eventos para que el programa realice las acciones requeridas, como puede ser la generación de la melodía.

Para crear el menú de "Ayuda" *ModusXXI* utiliza las siguientes clases de **`java.awt`**:

- **`java.awt.Menu`**. Instancia el objeto **`menuAyuda`** que contiene el objeto **`menuBarra`**.
- **`java.awt.MenuBar`**. Instancia el objeto **`menuBarra`** que contiene los objetos **`itemAyuda`**, **`itemAbout`** e **`itemNet`**.
- **`java.awt.MenuItem`**. Instancia los objetos **`itemAyuda`**, **`itemAbout`** e **`itemNet`** que funcionan como vínculos a los archivos que se encuentran en la carpeta de ayuda, a la clase **`AboutBox`** y a un URL en Internet respectivamente.

ModusXXI utiliza la biblioteca **javax.swing** para generar los componentes de la interfaz gráfica. Los objetos utilizados son:

- **javax.swing.Box** y **javax.swing.BoxLayout**. Estas clases definen el diseño del panel central de la ventana principal de *ModusXXI.*
- **javax.swing.ButtonGroup** y **javax.swing.JCheckBox**. Estas clases sirven para crear y agrupar objetos del tipo **CheckBox** (casillas de verificación).
- **javax.swing.JComboBox**. La clase **JComboBox** genera objetos del tipo ComboBox (selectores de opción).
- **javax.swing.JFrame**. La clase **JFrame** genera objetos del tipo **Frame** (cuadro).
- **javax.swing.JLabel**. La clase **JLabel** genera objetos del tipo **Label** (etiqueta).
- **javax.swing.JOptionPane**. La clase **JOptionPane** genera objetos del tipo **OptionPane** (ventana de diálogo).
- **javax.swing.JPanel**. La clase **JPanel** genera objetos del tipo **Panel** (panel).
- **javax.swing.JCheckBox**. La clase **JCheckBox** genera objetos del tipo **CheckBox** (casilla de verificación).
- **javax.swing.JTextField**. La clase **JTextField** genera objetos del tipo **TextField** (campo de texto).

Todos estos objetos son declarados en la clase **ModusXXI.java**, junto con los siguientes arreglos de tipo **string**: **capitulos**, **compases** y **notas**. Posteriormente son inicializados dentro del constructor y agregados al panel central; que es un objeto **Box** de la biblioteca **javax.swing**. Los eventos generados por el usuario son interceptados a través del método **public void actionPerformed(ActionEvent evt)**. Dependiendo de su evaluación estos eventos pueden ser tratados como excepciones, errores, alterando o manteniendo la configuración de la interfaz, a través de ventanas de diálogo. También sirven para recabar datos o para generar acciones como la creación de la melodía, su reproducción, etc.

3.3.4.3 La clase `AboutBox.java`

La clase **`AboutBox.java`** presenta la ventana "Acerca de *ModusXXI*", la cual contiene información general del programa como la versión, fecha de creación, propietario, desarrolladores, etc. El vínculo que usa esta clase se encuentra en el objeto **`itemAbout`.** Utiliza también las bibliotecas **`java.awt`** y **`javax.swing`**. Usa los métodos **`public void setVisible()`**, **`public void setCenteredLocation()`** y la clase **`AboutCloser`**.

3.3.4.4 La clase `OpenHelp.java`

La **clase `OpenHelp.java`** abre los archivos de la carpeta "Ayuda". Estos archivos se encuentran en formato HTML y contienen el Manual de Usuario. Aparte de su constructor, esta clase contiene tres métodos boléanos que reconocen en qué tipo de sistema operativo se está corriendo *ModusXXI*: **`isWindows()`**, **`isMac()`**, **`isLinux()`**. Cada uno de estos métodos determina el tipo de sistema operativo. Estos métodos son muy importantes, pues una vez reconocido el sistema operativo, indican al constructor el comando de ejecución del navegador de default para poder visualizar los archivos HTML de la carpeta de "Ayuda". **`OpenHelp.java`** utiliza la clase **`java.io.IOException`**. para atender excepciones.

3.3.4.5 Carpeta "docs"

La carpeta "docs" contiene tres carpetas: la carpeta "Ayuda", la carpeta "Instalacion" y la carpeta "ManualDeUsuario". La carpeta "Ayuda" contiene los archivos en formato HTML, que serán abiertos desde el objeto **`itemAyuda`** de *ModusXXI*. Dado que desde la programación de la clase **`OpenHelp.java`** está definida la ruta de la carpeta "Ayuda", debe permanecer ésta siempre dentro de la carpeta "docs" para poder ser localizada. Asimismo, la carpeta "docs" debe estar siempre dentro de la carpeta "ModusXXI". Las carpeta "Instalación" y "Manual de Usuario" serán tratadas en el siguiente capítulo.

4. Resultados

4.1 Archivos que integran la aplicación *ModusXXI*

La aplicación *ModusXXI* está conformada por una carpeta llamada “ModusXXI”. La carpeta “ModusXXI” posee tres archivos y una carpeta:

- ***ModusXXI***. Este es el archivo de la aplicación en formato .app para sistemas operativos Macintosh OsX. Sin ser una aplicación nativa, posee las propiedades de las aplicaciones nativas de Macintosh. Otra ventaja que ofrece este archivo a los usuarios de Macintosh es que no requiere la carpeta “docs” para visualizar la ayuda. Para correr este archivo, con toda su funcionalidad, sólo es necesario copiarlo a la carpeta de aplicaciones (Requiere la previa instalación de JVM 1.4 o superior).
- ***ModusXXI.jar***. Este es el archivo de la aplicación en formato .jar. Este archivo puede ejecutarse en cualquier sistema operativo que soporte la máquina virtual de java (jvm 1.4 o superior). Para su correcto funcionamiento es necesario copiar la carpeta “ModusXXI” en la carpeta de aplicaciones. Este archivo también puede ejecutarse en el sistema operativo Macintosh OsX.
- **El archivo de texto README.txt.** Este archivo contiene las últimas actualizaciones del programa, así como informaciones generales para el usuario.
- **La carpeta “docs”**. Esta carpeta contiene la documentación del programa. Posee tres carpetas:
  - **Ayuda**. Carpeta que contiene en formato HTML los archivos que conforman la ayuda del programa.
  - **Instalación**. Contiene el documento “Instalación” en formato PDF y formato HTML.
  - **ManualDeUsuario**. Contiene el documento “ManualDeUsuario” en formato PDF y formato HTML.

4.2 Instalación de la aplicación

Requerimientos del sistema:

- Java Virtual Maschine (JVM) 1.4 o superior
- Sistema operativo Microsoft Windows, Mac OsX, Linux, Solaris o cualquier otro con soporte de la máquina virtual de java.
- Tarjeta de audio con sintetizador interno que de soporte GeneralMIDI.

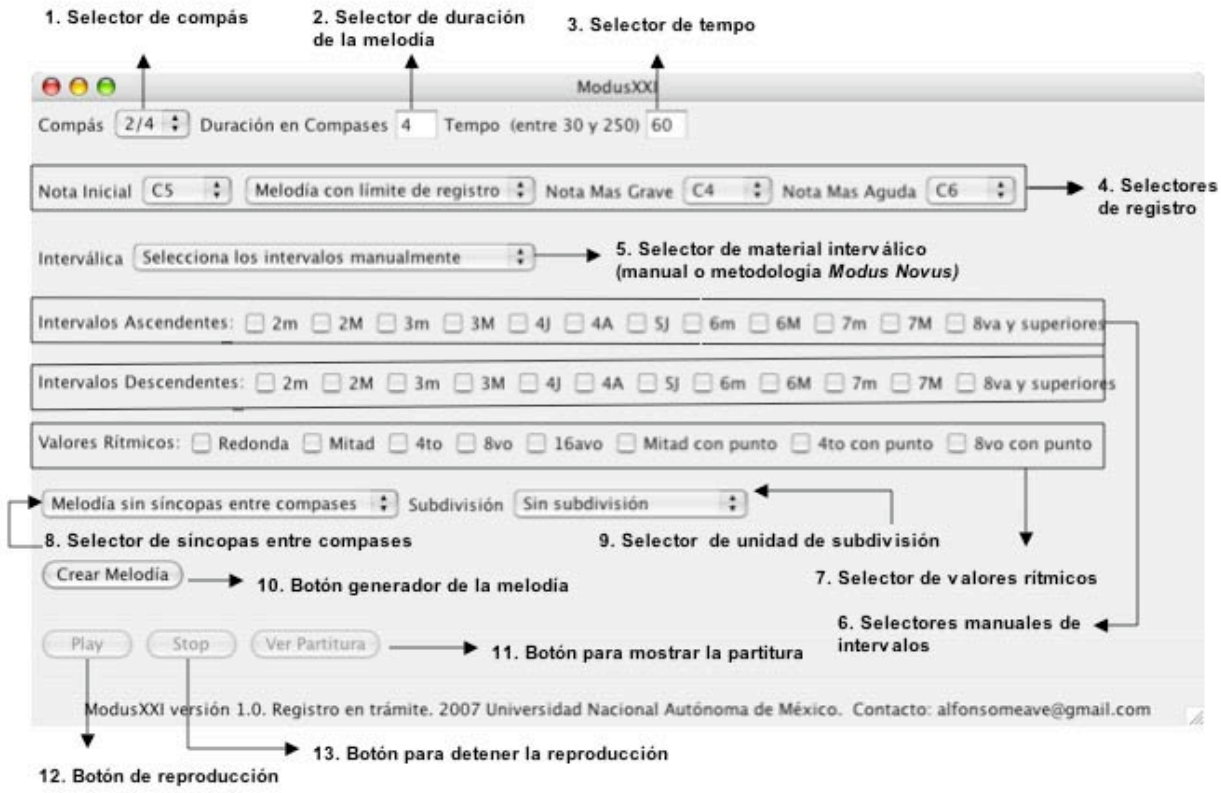
Instalación:

- Verificar si está instalada la Máquina Virtual de Java (JVM) 1.4 o superior. Si no está instalada, se puede descargar desde esta dirección electrónica: http://java.sun.com/j2se/1.4.2/. Instalar el programa siguiendo las instrucciones del fabricante.
- Descargar el archivo *ModusXXI.zip* en la computadora.
- Descomprimir el archivo *ModusXXI.zip* haciendo doble clic sobre él.
- Copiar la carpeta *ModusXXI* en la carpeta de aplicaciones.
- Reiniciar la computadora.

4.3 Elementos que componen *ModusXXI*

La aplicación de cómputo *ModusXXI* se compone de dos ventanas y un menú "Ayuda". La ventana principal sirve para la entrada de datos y la generación de la melodía. La ventana secundaria muestra la melodía en notación musical y es presentada al seleccionar el botón "Ver Partitura". El menú "Ayuda" se compone de tres opciones de selección: "Acerca de ModusXXI", "Ayuda" y "WebSite". Se presentan a continuación cada uno de estos elementos.

4.3.1 La ventana principal *ModusXXI*



La ventana principal de *ModusXXI* está integrada por nueve páneles. Los primeros siete paneles sirven para establecer los parámetros de generación del material melódico. Los paneles ocho y nueve son botones que generan eventos.

Panel uno:

1. *Selector de compás.* ComboBox *Compás*.
2. *Selector de duración de la melodía.* Campo de texto *Número de compases*.
3. *Selector de tempo.* Campo de texto *Tempo.*

Panel dos: *Selectores de registro tonal*

4.1 ComboBox *Nota Inicial*.

4.2 ComboBox *Melodía con límite de registro/Melodía sin límite de registro*.

4.3 ComboBox *Nota más grave.*

4.4 ComboBox *Nota más aguda*.

Panel tres: *Selector de material interválico*

5. ComboBox *Interválica*.

Panel cuatro: *Selectores manuales de intervalos*

6. Casillas de verificación *Intervalos ascendentes*.

Panel cinco: *Selectores manuales de intervalos*

6. Casillas de verificación *Intervalos descendentes*.

Panel seis: *Selectores de valores rítmicos*

7. Casillas de verificación *Valores Rítmicos*

Panel siete:

8. *Selector del control de síncopas entre compases*. ComboBox *Evitar síncopa entre compases/Permitir síncopa entre compases*.
9. *Selector de unidad de subdivisión*. ComboBox *Sin subdivisión/Subdivisión en cuartos/Subdivisión en octavos.*

Panel ocho:

10. *Botón generador de la melodía*. Botón *Crear melodía*. Con los parámetros definidos en los paneles anteriores este botón va a llamar a la clase `GeneradorDeEjercicios` para crear una nueva melodía.

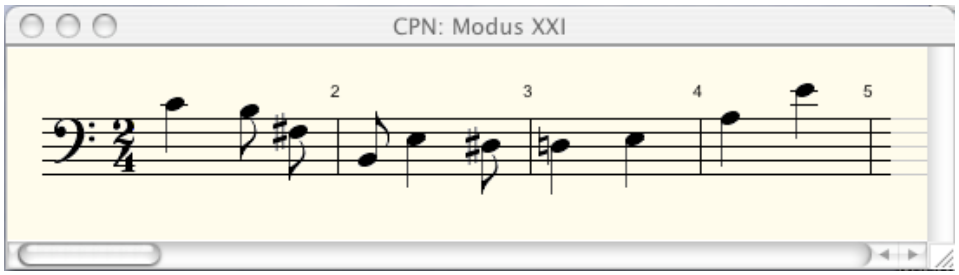
Panel nueve:

11. *Botón para mostrar la partitura*. Botón *Ver partitura*
12. *Botón de reproducción*. Botón *Play*.
13. *Botón para detener la reproducción*. Botón *Stop (parada)*.

Tabla: Paneles y selectores

Panel	Selectores	Selectores	Selectores
Panel 1	1. Selector de compás	2. Selector de duración de la melodía en número de compases	3. Selector de tempo
Panel 2	4. Selectores de registro tonal		
Panel 3	5. Selector de material interválico		
Panel 4	6. Selectores manuales de intervalos (ascendentes)		
Panel 5	6. Selectores manuales de intervalos (descendentes)		
Panel 6	7. Selectores de valores rítmicos		
Panel 7	8. Selector del control de síncopas entre compases	9. Selector de unidad de subdivisión	
Panel 8	10. Botón generador de melodía		
Panel 9	11. Botón de reproducción	12. Botón para detener la reproducción	13. Botón para mostrar la partitura

4.3.2 La ventana con la partitura



Esta ventana se abre al hacer clic en el botón *Ver partitura*, y es una representación gráfica en notación musical de la melodía generada por *ModusXXI*. Esta ventana posee cuatro menús proporcionados por la biblioteca *jMusic*: File, Tools, Play y View. A continuación se describen las funciones más relevantes de estos menús para el usuario de *ModusXXI*.

- Guardar la melodía como archivo MIDI: selecciona *File* y después selecciona *Save as MIDI File*.
- Cerrar la ventana: selecciona *File* y después selecciona *close*.
- Cambiar instrumento GeneralMidi (*ModusXXI* genera por default las melodías con el *patch* GrandPiano de GeneralMidi), volumen o tempo: selecciona *Tools* y después selecciona *Set Parameters*. Aparecerá una nueva ventana, allí hay que seleccionar el parámetro a modificar, hacer clic en *Apply* y después en *Close*.
- Reproducir desde la ventana de la partitura: selecciona *Play* y después selecciona *Play All*.
- Detener la reproducción: : selecciona *Play* y después selecciona *Stop Playback*.

Si se desea más información respecto a la biblioteca *jmusic* y su manejo se puede remitir al siguiente URL: http://jmusic.ci.qut.edu.au

4.3.3 El menú “Ayuda”

El menú “Ayuda” está compuesto por tres opciones:

- *Acerca de ModusXXI*. Al activarse esta opción se presenta la siguiente ventana con información general sobre el programa:

- Ayuda. Esta opción establece un vínculo con la carpeta “Ayuda” la cual contiene los archivos del Manual de Usuario.
- WebSite. Esta opción establece un vínculo con un URL en Internet (página web del proyecto, en construcción).

4.4 Funcionamiento de *ModusXXI*

4.4.1 Generación de material melódico

Para generar melodías con *ModusXXI* se requieren los siguientes pasos mínimos:

1. Hacer doble clic en el archivo *ModusXXI.jar* (*ModusXXI* para usuarios Macintosh) . Aparecerá la ventana principal de *ModusXXI*.
2. Hacer clic en el selector de **Compás** y escoger el tipo deseado. 2/4 está activado por default.
3. Introducir en el campo **Duración en compases** el número de compases que se desea dure la melodía. Cuatro (**4**) compases aparecen por default.
4. Introducir en el campo **Tempo** la velocidad de la melodía a generar. Cuarto (negra) = 60 aparece por default.
5. Hacer clic en el selector **Interválica** y seleccionar **Capítulo 1 (2m, 2M , 4J)**, o algún otro capítulo que se desee.
6. Hacer clic en algunas de las casillas de verificación del selector de **Valores Rítmicos**. Por ejemplo en **4to** y **8vo**. Una vez hecho clic, las casillas aparecerán activadas.
7. Presionar el botón "**Crear Melodía**".
8. Presionar el botón "**Play**". Se deberá escuchar una melodía.
9. Presionar el botón "**Ver Partitura**" para ver la melodía en notación musical.

4.4.2 Estructura de control

1. **Selector de compás.** Activa el compás en que se desea generar la melodía. Las opciones van de 2/4, 3/4, 4/4,…hasta 9/4.
2. **Selector de duración de la melodía.** Aquí debe introducirse un número entero de compases que se desea que dure la melodía. Por ejemplo, si se desea que dure la melodía dos compases se escribe 2, si se desea que dure cinco se escribe 5, etc. *ModusXXI* es capaz de generar melodías con una duración de hasta 67 cuartos o figuras equivalentes. (Límite impuesto por la biblioteca *jMusic*.)
3. **Selector de tempo.** Aquí se debe introducir un valor numérico entero que defina la velocidad de la melodía que va a generar o reproducir *ModusXXI*. El valor numérico del tempo debe encontrarse entre 30 y 250 (pulsos por segundo). Tanto en la generación como en la reproducción, puede alterarse el tempo en *ModusXXI*.
4. **Selectores de registro tonal.** Los selectores de registro tonal comprenden cuatro controles tipo ComboBox:

a) **Nota Inicial.** Escoge la nota que se desea al inicio de la melodía a generar por *ModusXXI*. La selección va de Do0 a Sol10 (Registro que permite el protocolo MIDI, donde Do0 equivale a 0 y Sol10 a 127). Do5 está activado por default.

b) **Melodía con límite de registro o Melodía sin límite de registro. Melodía sin límite de registro** desactiva la **nota más grave** y la **nota más aguda**, permitiendo que los límites de registros sean Do0 y Sol10. **Melodía con límite de registro** activa la **nota más grave** y la **nota más aguda**.

c) **Nota Más Grave.** Selecciona la nota límite en el registro grave para la generación de la melodía. La selección va de Do0 a Sol10.

d) **Nota Más Aguda.** Selecciona la nota límite en el registro agudo para la generación de la melodía. La selección va de Do0 a Sol10.

5. **Selector de material interválico de acuerdo a la metodología *Modus Novus*.** Este selector posee tres grupos básicos de activación del material interválico:

   a) **"Selecciona los intervalos manualmente".** Al escoger esta opción del selector, el usuario debe activar manualmente las casillas de verificación de los **"Selectores manuales de intervalos"**. Esta opción ofrece también la selección de la dirección, ascendente o descendente, del intervalo. Por ejemplo, se puede seleccionar únicamente el intervalo de 5J (quinta justa) ascendente. Si se desea que el intervalo de 5J se presente tanto ascendente como descendente, es necesario activar ambas casillas de verificación.

   b) **"Modus Novus".** Esta opción requiere la selección específica de algún capítulo del *Modus Novus*. Al seleccionar algún capítulo y pulsar el botón **Crear Melodía** se activarán automáticamente las casillas de verificación de los intervalos, ascendentes y descendentes, correspondientes al capítulo del *Modus Novus* que se desee practicar. Por ejemplo, al seleccionar **Capítulo 1** los intervalos de segunda menor, segunda mayor y cuarta justa (ascendentes y descendentes) serán activados automáticamente.

   c) **"Sugerencia de estudio"**. Esta opción sirve de estudio preparatorio para los capítulos del *Modus Novus*. A diferencia del material generado con las opciones de "**Modus Novus",** las opciones de **"Sugerencia de estudio"** crean melodías únicamente con el material interválico nuevo que contiene cada capítulo del

Modus Novus; unido a través de intervalos de segunda menor y mayor. Por ejemplo en **3 (2m, 2M, 3m, 3M)** el material de estudio son los intervalos de tercera menor y tercera mayor. Las melodías serán generadas utilizando únicamente dicho material nuevo (3m y 3M en este caso) unido por intervalos de segunda menor y mayor. En cambio, al seleccionar **Capítulo 3** también van a ser seleccionados los intervalos de los capítulos anteriores (en este caso cuarta justa, quinta justa, además de segunda menor y segunda mayor). Esta opción permite un estudio más detallado del material nuevo que presenta cada capítulo del *Modus Novus.*

6. **Selectores manuales de intervalos.** Son casillas de verificación que sirven para seleccionar manualmente los intervalos en dirección tanto ascendente como descendente. Los intervalos a seleccionar van de la segunda menor a la séptima mayor. Existe además una casilla de verificación, ascendente y descendente, que reúne un grupo de intervalos iguales o mayores a la octava. El registro límite de dichos intervalos es de tres octavas.
7. **Selectores de valores rítmicos.** Activan los valores rítmicos que se desean en la melodía. Dichos valores son redonda, mitad, cuarto, octavo, dieciseisavo, mitad con punto, cuarto con punto y octavo con punto.
8. **Selector para permitir síncopas entre compases ó evitar con síncopas entre compases.** La generación de figuras rítmicas en *ModusXXI* es también un proceso aleatorio, el cual tiene como límites de generación y selección de las figuras rítmicas a:
    a) **La duración total en compases de la melodía.** *ModusXXI* escoge valores rítmicos de forma aleatoria hasta llenar el espacio total de compases definidos por el usuario. Si el usuario selecciona **Permitir síncopas entre compases**, *ModusXXI* va a hacer su selección de valores rítmicos sin tomar en cuenta a la unidad de compás. La consecuencia de esto va a ser la generación de material melódico posiblemente con síncopas entre compases. La selección de **Permitir síncopas entre compases** va a desactivar el **selector de unidad de subdivisión**.
    b) **La duración en cuartos de cada compás (Unidad de medida).** Al activar **Evitar síncopas entre compases** *ModusXXI* selecciona las figuras rítmicas tomando en cuenta tanto la duración total en compases como la unidad de

medida del compás; i.e. dos para 2/4, tres para 3/4, cuatro para 4/4, etc. Como resultado las síncopas entre compases son eliminadas y consecuentemente las melodías tendrán siempre compases téticos. Por default está activada la función **Evitar síncopas entre compases.**

c) **El número de pulsos en cuartos o octavos de cada compás (unidad de subdivisión).** Al seleccionar el material rítmico *ModusXXI* también puede tomar en cuenta, aparte del total de compases y la unidad de medida, la unidad de tiempo o pulso. La consecuencia va a ser no sólo la eliminación de síncopas dentro del compás, sino también la imposibilidad de seleccionar valores superiores a la unidad de subdivisión seleccionada. Esta función va a depender del **selector de unidad de subdivisión.**

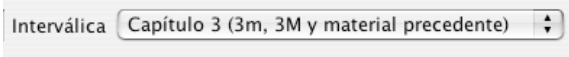
9. **Selector de unidad de subdivisión.** Este selector ofrece tres posibles activaciones:
   a) **Sin subdivisión.** Activada por default, toma como parámetro de selección del material rítmico a la unidad de compás. Consecuencia de su activación es la generación de material rítmico tético, pero con posibilidad de síncopas dentro del compás.
   b) **Subdivisión en cuartos.** Toma como parámetro de selección del material rítmico al cuarto. La activación de esta opción genera secuencias de material rítmico de cuartos, octavos y dieciseisavos. Consecuentemente, se imposibilita la selección de valores rítmicos superiores al cuarto.
   c) **Subdivisión en octavos.** Toma como parámetro de selección del material rítmico al octavo. La activación de esta opción genera secuencias de material rítmico de octavos y dieciseisavos. Consecuentemente, se imposibilita la selección de valores rítmicos superiores al octavo.
10. **Botón generador de la melodía (“Crear Melodía”).** Cada vez que se oprima este botón se va a generar una nueva melodía.
11. **Botón para mostrar la partitura (“Ver partitura”).** Al oprimir este botón va a aparecer en la pantalla una nueva ventana con la notación musical de la melodía que se generó.
12. **Botón de reproducción (“Play”).** Al oprimir este botón va a reproducirse la melodía generada.

13. **Botón para detener la reproducción (“Stop”).** Al oprimir este botón se va a detener la reproducción de la melodía.

4.4.2.1 El material interválico

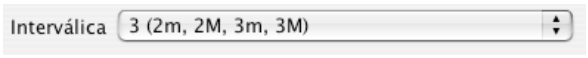
El material interválico es el punto de relación entre el *Modus Novus y el ModusXXI*. Resulta necesario hacer una exposición detallada del funcionamiento respectivo en la aplicación. *ModusXXI* posee en un selector o comboBox etiquetado como **Interválica.** Por default en este selector está activada la opción “**Selecciona los intervalos manualmente”**. Es necesario que esta opción esté activada si se desea seleccionar los intervalos desde las casillas de verificación. Si por el contrario, se desea hacer un estudio siguiendo la metodología *Modus Novus* es necesario hacer clic en el comboBox de **Interválica** y seleccionar alguna de las opciones de estudio que ofrece la opción **Modus Novus (Selecciona algún capítulo)**.

Por ejemplo:



Si se selecciona esta opción *ModusXXI* generará melodías con los intervalos de estudio del Capítulo 3 del *Modus Novus*, es decir 2m, 2M, 3m, 3M, 4J y 5J. Lo mismo sucederá si se escoge cualquiera de las opciones intitulada **Capítulo...**

La opción **Sugerencia de estudio** es una propuesta metodológica nuestra. Comprende trece opciones distintas de estudio las cuales sirven de estudio preparatorio para cada uno de los capítulos del *Modus Novus.* A diferencia de las opciones que ofrece la selección **Capítulo…**, **las Sugerencias de estudio** generan melodías únicamente con el material interválico nuevo que presenta cada capítulo del *Modus Novus*, unido por intervalos de segunda mayor y menor. Por ejemplo, la opción

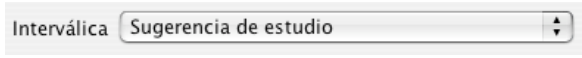


va a generar melodías únicamente con intervalos de 2m, 2M 3m y 3M. Esta opción elimina los materiales precedentes; en este caso la 4J y la 5J. Antes de generar melodías con la opción **Capítulo 3 (3m, 3M y material precedente)** se puede hacer un estudio previo usando la opción **3 (2m, 2M, 3m y 3M)**. (Para más información consulte la tabla **Grupos básicos de selección de material interválico en el *ModusXXI*** que se encuentra más abajo.)

Los capítulos 4, 8 y 11 del *Modus Novus* presentan ejemplos melódicos de la literatura musical. Por esta razón no son considerados dentro de las opciones que ofrece

.

Las opciones 4, 8 y 11 de



ofrecen síntesis de los materiales precedentes. La opción 13 permite la generación de material melódico con intervalos iguales o superiores a la octava. (Para más información consulte la tabla **Grupos básicos de selección de material interválico en el *ModusXXI*** que se encuentra más abajo.)

Para generar material interválico de manera libre sólo es necesario seleccionar la opción

y activar manualmente las casillas de verificación de los intervalos:

Esta opción permite asimismo la selección de la dirección (ascendente o descendente) de los intervalos.

4.4.2.2 Tabla: Grupos básicos de selección de material interválico en el *ModusXXI*

Presentamos a continuación una tabla que resume las diferentes opciones del selector de material interválico.

“Selecciona los intervalos manualmente”	**“Modus Novus”**	**“Sugerencia de estudio”**
El usuario debe activar manualmente las casillas de verificación de los selectores manuales de intervalos.	Capítulo 1 (2m, 2M, 4J)	1 (2m, 2M, 4J)
Atención: ¡Esta opción debe estar activada si se desea usar los selectores manuales de intervalos!	Capítulo 2 (5J y material precedente)	2 (2m, 2M, 5J)
	Capítulo 3 (3m, 3M y material precedente)	3 (2m, 2M, 3m, 3M)
		4 (2m, 2M, 3m, 3M, 4J, 5J) Resumen de intervalos de los capítulos 1, 2 y 3 del Modus Novus
	Capítulo 5 (Tritono y material precedente)	5 (2m, 2M, Tritono)
	Capítulo 6 (6m y material precedente)	6 (2m, 2M, 6m)
	Capítulo 7 (6M y material precedente)	7 (2m, 2M, 6M)
		8 (2m, 2M, 4A, 6m, 6M) Resumen de intervalos de los capítulos 5, 6 y 7 del Modus Novus
	Capítulo 9 (7m y material precedente)	9 (2m, 2M, 7m)
	Capítulo 10 (7M y material precedente)	10 (2m, 2M, 7M)
		11 (2m, 2M, 4A, 6m, 6M, 7m, 7M) Resumen de intervalos de los capítulos 5, 6, 7, 9 y 10 del Modus Novus
	Capítulo 12 (Weitmelodik. Octava e intervalos superiores)	12 De 2m a 7M Resumen de intervalos de los capítulos 1 al 10 del Modus Novus
		13 Intervalos de octava y superiores (Sin tomar en cuenta los intervalos inferiores a la octava)

4.4.3 Prevención de errores

Como se mencionó en el capítulo anterior (3.Desarrollo), la prevención y resolución de errores es uno de los principales aspectos de la programación. Para prevenir, atrapar y resolver errores *ModusXXI* realiza las siguientes acciones:

1. Poseer valores de default

ModusXXI cuenta con los siguientes valores de default:

- Tipo de compás: 2/4
- Número de compases (duración): 4
- Tempo: 60
- Nota inicial: C5 (do índice cinco)
- Melodía con límite de registro. Esta opción habilita:
  - Nota más grave: C4 (do índice cuatro)
  - Nota más aguda: C6 (do índice seis)
- Selecciona los intervalos manualmente
- Evitar síncopas entre compases. Esta opción habilita:
  - Subdivisión
- El botón “Crear Melodía” está habilitado
- Los botones “Play”, “Stop” y “Ver Partitura” están deshabilitados.

Para generar una melodía por primera vez el usuario necesita únicamente seleccionar la interválica y los valores rítmicos.

2. Deshabilitar botones

ModusXXI deshabilita las siguientes opciones:

- “Play”, “Stop” y “Ver Partitura” en caso de no estar creada una melodía.
- Selectores “Nota más grave” y “Nota más aguda” si está seleccionada la opción “Melodía sin límite de registro”.
- Selector “Subdivisión” si está seleccionado “Permitir síncopas entre compases”.

3. Ignorar cambios que provoquen errores, regresando el valor anterior

ModusXXI atrapa las acciones erróneas del usuario que no tengan consecuencias en la función a realizar, regresando simplemente el valor anterior. Casos:

- Si se introduce en el campo “Número de compases” un número negativo o un carácter no numérico, son ignorados y se devuelve el valor anterior contenido en esta variable.

4. Presentar ventanas de diálogo

Las ventanas de diálogo ofrecen información sobre acciones erróneas o alteraciones que pueden tener consecuencias no deseadas por el usuario. En *ModusXXI* se usan dos tipos de ventanas de diálogo:

- **De dos opciones**: *Sí* y *No*. Informa al usuario de las consecuencias de cierta acción. De acuerdo a la decisión del usuario se ejecuta un tipo acción. Casos:
  - Si se selecciona cualquier opción de la ventana principal cuando se tiene creada una melodía se abre una ventana de diálogo con el siguiente mensaje: *"Está cambiando la configuración. ¿Desea generar una nueva melodía? (La configuración anterior se perderá)."* Esta ventana posee dos opciones de selección: *Sí* y *No*. La opción *No* está seleccionada por default. Si esta opción es ratificada por el usuario, la configuración anterior es devuelta. Si el usuario selecciona *Sí* entonces se desactivan los botones “Play”, “Stop” y “Ver Partitura” perdiéndose la melodía previa, pero permitiéndose al usuario crear una nueva configuración.
- **De una opción**. Informa al usuario sobre un error o excepción. Regresa el valor anterior. Casos:

Ventanas de diálogo que pueden aparecer al presionar el botón “Play”:

- Si se introduce en el campo “Tempo” un número mayor a 250 y menor a 30 se abre una ventana de diálogo con el siguiente mensaje:*"El tempo requiere un número entre 30 y 250"* y regresa el valor anterior del tempo.
- Si se introduce en el campo “Tempo” un carácter no numérico se abre una ventana de diálogo con el siguiente mensaje: *"El Tempo requiere un valor numérico"* y regresa el valor anterior del tempo.

Ventanas de diálogo que pueden aparecer al presionar el botón “Crear”:

- Si se introduce en el campo “Número de compases” un carácter no numérico se abre una ventana de diálogo con el siguiente mensaje: *"El número de compases requiere un valor numérico"*.
- Si se introduce en el campo “Duración en compases” un número negativo se abre una ventana de diálogo con el siguiente mensaje: *"El número de compases requiere un valor numérico positivo"*.
- Si se introduce en el campo “Duración en compases” un número entero mayor a 30[1] se abre una ventana de diálogo con el siguiente mensaje: *“ModusXXI genera melodías con una duración máxima de 67 cuartos o equivalente"*.
- Si se introduce en el campo “Tempo” un número mayor a 250 y menor a 30 se abre una ventana de diálogo con el siguiente mensaje:*"El tempo requiere un número entre 30 y 250"* y regresa el valor anterior del tempo.
- Si se introduce en el campo “Tempo” un carácter no numérico se abre una ventana de diálogo con el siguiente mensaje: *"El Tempo requiere un valor numérico"* y regresa el valor anterior del tempo.
- Si se seleccionan figuras rítmicas superiores al numerador del compás, por ejemplo compás de 2/4 con redondas se abre una ventana de diálogo con el siguiente mensaje: *"Las figuras rítmicas son incompatibles con el tipo de compás"*.
- Si se seleccionan intervalos superiores al registro se abre una ventana de diálogo con el siguiente mensaje: *"Los intervalos son incompatibles con el registro"*.
- Si la “nota más grave” está por encima de la “nota más aguda” se abre una ventana de diálogo con el siguiente mensaje: *"La nota más grave está por arriba de la nota más aguda"*.
- Si la “nota más aguda” está por debajo de la “nota más grave” se abre una ventana de diálogo con el siguiente mensaje: *"La nota más aguda está por debajo de la nota más grave"*.

[1] La biblioteca *jMusic* permite crear melodías con una duración máxima de 67 cuartos o equivalente. El número máximo de compases con esta condición es de 30 en compás de 2/4.

- Si la “nota inicial” está por debajo de la “nota más grave” o por encima de la “nota más aguda” se abre una ventana de diálogo con el siguiente mensaje: *"La nota inicial está por debajo de la nota más grave o por arriba de la nota más aguda"*.
- Si no es seleccionado algún intervalo o alguna opción del selector de interválica se abre una ventana de diálogo con el siguiente mensaje: *“No ha sido seleccionada la interválica”*.
- Si no ha sido seleccionada alguna figura rítmica se abre una ventana de diálogo con el siguiente mensaje: *“No ha sido seleccionados los valores rítmicos”*.
- Si es generada una melodía se abre una ventana de diálogo con el siguiente mensaje: *“Ha sido creada una melodía. Selecciona el botón Play para escucharla ó el botón Ver partitura"*.

4.4.4 Documentación

ModusXXI cuenta con una carpeta titulada “docs” la cual contiene la documentación del programa. Dicha carpeta está integrada por tres carpetas: Ayuda, Instalación y ManualDeUsuario.

La carpeta Ayuda

Contiene en formato HTML los archivos que conforman el manual de usuario del programa. Para acceder a ella hay que seleccionar el ítem “Ayuda” del menú “Ayuda” de la ventana principal de *ModusXXI*. Al seleccionar este ítem *ModusXXI* correrá el navegador (Browser) de default del sistema para poder visualizar la ayuda. Apareciendo la siguiente ventana:

Esta ventana cuenta con ocho menús con información para el usuario. Si se desea conocer la información del manual de usuario consulte el apéndice.

La carpeta Instalación.

Contiene el documento “Instalacion”. Este es el manual de instalación y se encuentra en dos formatos: PDF y HTML. El formato PDF puede ser impreso para contar con el documento en formato papel. El archivo HTML es un formato electrónico y puede ser visualizado a través de algún navegador. Puede consultarse la versión en papel en el apéndice de la presente tesis.

La carpeta ManualDeUsuario.

Contiene el documento “ManualDeUsario”. Este contiene el manual de usuario y se encuentra en dos formatos: PDF y HTML. El formato PDF puede ser impreso para contar con el documento en formato papel. El archivo HTML es un formato electrónico y puede ser visualizado a través de algún navegador. Puede consultarse la versión en papel en el apéndice de la presente tesis.

4.5 Pruebas : Evaluación de uso y calidad de la aplicación de cómputo *ModusXXI versión 1.0*

Metodología de evaluación

Para realizar esta evaluación de uso y calidad hemos usado el sistema de evaluación MOS (*Mean Opinion Score*)[2], el cual emplea un rango numérico de cinco puntos; donde el número menor denota baja calidad y el número mayor alta calidad.

MOS	Calidad
4	Excelente
3	Buena
2	Satisfactoria
1	Pobre
0	Mala

Se elaboró un cuestionario de nueve preguntas sobre la aplicación *ModusXXI*, para su evaluación de acuerdo al puntaje MOS:

Excelente ☐ Buena ☐ Satisfactoria ☐ Pobre ☐ Mala ☐.

Las preguntas son las siguientes:

1. Consideras la claridad de la interfaz gráfica de *ModusXXI*:
2. Consideras la facilidad de manejo de *ModusXXI*:
3. Consideras la información que ofrece el manual de usuario:
4. En términos de efectividad de uso y eficiencia, consideras el funcionamiento de *ModusXXI*:
5. Consideras la representación de los conceptos musicales (intervalos, valores rítmicos, etc.) a través de los componentes de la interfaz gráfica (botones, casillas de verificación, etc.) de *ModusXXI*:
6. Para el estudio del *Modus Novus* y la música no tonal, *ModusXXI* puede ser una herramienta:
7. Para mejorar tu capacidad auditiva, Consideras que *ModusXXI* puede ser:

[2] http://www.itu.int/rec/T-REC-P.800.1-200607-I/en

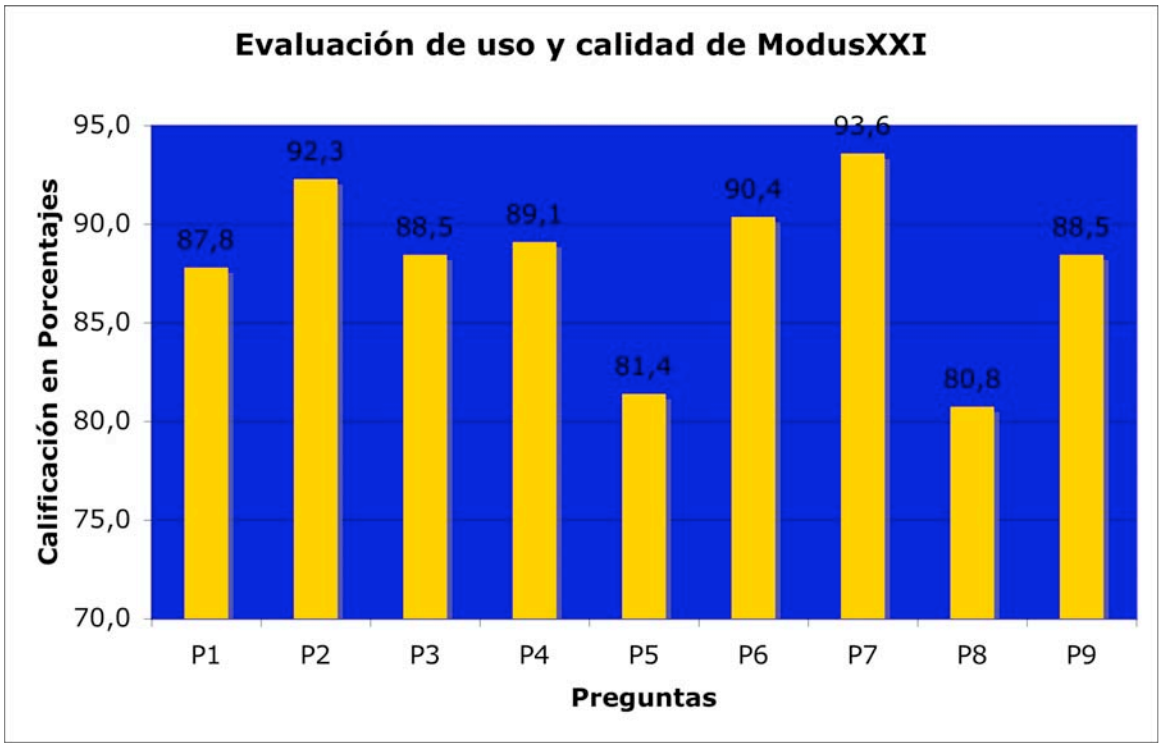
8. Como herramienta didáctica para enseñanza en grupo, consideras que *ModusXXI* puede ser:
9. Como ayuda para un estudio independiente del *Modus Novus* y la música no tonal consideras que *ModusXXI* puede ser:

Asimismo, se integró el siguiente punto para que los usuarios pudieran escribir opiniones:

10. Si tienes algún comentario o sugerencia sobre *ModusXXI* escribelo por favor al reverso de esta hoja.

4.6 Resultados de la evaluación

La aplicación fue evaluada por 39 usuarios obteniéndose los siguientes resultados:



En porcentaje de evaluación, *ModusXXI* fue calificado de la siguiente manera:

Pregunta	**Calificación en Porcentaje de *ModusXXI***
P7. Para mejorar tu capacidad auditiva	93.6
P2. Facilidad de manejo	92,3
P6. Para el estudio del *Modus Novus* y la música no tonal, *ModusXXI* puede ser una herramienta	90,4
P4. Efectividad de uso y eficiencia	89,1
P3. Facilidad de manejo	88,5
P9 Ayuda para un estudio independiente del *Modus Novus* y la música no tonal	88,5
P1. Claridad de la interfaz gráfica	87,8
P5. Representación de los conceptos musicales a través de los componentes de la interfaz	81,4
P8. Herramienta didáctica para enseñanza en grupo	80,8

El porcentaje más alto (93,6) lo obtuvo la pregunta siete (P7). En su mayoría, los usuarios consideran que *ModusXXI* puede mejorar su capacidad auditiva. El puntaje más bajo lo obtuvo la pregunta ocho (P8) (80,8). Parece que para un porcentaje considerable de usuarios, *ModusXXI* no es la mejor herramienta didáctica en grupo. La facilidad de manejo (P2) y el uso de *ModusXXI* para un estudio de la música no tonal (P6) fueron también evaluados positivamente por los usuarios (92,3 y 90,4 respectivamente); mientras que la representación de los conceptos musicales a través de los componentes de la interfaz (P5) obtuvo una calificación relativamente baja (81,4). Las preguntas uno, tres cuatro y nueve (P1, P3, P4 y P9) se mantuvieron dentro de un rango de evaluación aceptable (89.1 y 88,5). Vale la pena señalar que estos porcentajes obtenidos superiores al 80%, corresponden a la categoría “excelente” del MOS.

Comentarios de los usuarios encuestados

Los comentarios realizados por los usuarios en el cuestionario de evaluación han sido agrupados en tres categorías: Errores del programa, Diseño y sugerencia de Nuevas funciones en versiones posteriores.

Comentarios de los usuarios sobre *ModusXXI*

Errores del programa	**Diseño de la interfaz de usuario**	**Nuevas funciones para versiones posteriores**
Ausencia de alteraciones en bemoles	Diseñar una interfaz más interactiva	Generar intervalos armónicos y acordes
Representación de las síncopas con una sola nota.	Mejorar el diseño de la interfaz gráfica de usuario	Generar melodías tonales
	Mejorar la calidad visual de gráficos en el archivo de ayuda	Posibilitar silencios en las melodías
		Posibilitar la generación de melodías en compases compuestos

Errores del Programa

- Ausencia de alteraciones en bemoles. Esta observación es recurrente. En *ModusXXI* las notas alteradas son siempre sostenidos. Esto está determinado por la biblioteca *jMusic*, por lo que en principio, una posible modificación implicaría una corrección en la biblioteca *jMusic*, más que un cambio en *ModusXXI*. Algunos usuarios consideran el uso de enarmonías (i.e. la sostenido en lugar de si bemol) confusas para la lectura o "faltas de ortografía musical". No obstante, dado el contexto atonal de *ModusXXI*, las enarmonías son completamente permisibles. Más aún, en el propio *Modus Novus*[3] encontramos constantemente enarmonías. Edlund escribe intervalos de tercera disminuida para describir segundas mayores, o terceras aumentadas para cuartas justas. Si bien la implementación de alteraciones en bemoles en una versión nueva de *ModusXXI* es una buena propuesta, consideramos

[3]Edlund Lars, *Modus Novus*, pag. 19.

que la actual notación, con alteraciones únicamente en sostenidos, no genera una escritura errónea. La confusión provocada por las enarmonías es resultado de la naturaleza de los sistemas de notación y nomenclatura musicales tradicionales; así como, posiblemente, de la falta de entrenamiento del usuario respecto a éstas.

- Representación de las síncopas con una sola nota. Las síncopas son representadas en *ModusXXI* por una nota completa (no por dos notas unidas por una ligadura). Consecuentemente la barra de compás es eliminada en la representación gráfica. Este es también un problema generado por la biblioteca *jMusic*; ajeno a las funciones propias de *ModusXXI*. No obstante, es un problema importante que debe resolverse en futuras versiones del programa.

Diseño de la interfaz de usuario

- Diseñar una interfaz interactiva. Algunos usuarios consideran que una interfaz más interactiva, que permita, por ejemplo, funciones de escritura de la melodía sobre la pantalla, evaluación de la propuesta del usuario, etc., sería más atractiva y motivadora. El desarrollo de este tipo de funciones debe ser considerado para versiones nuevas del programa.
- Mejorar el diseño de la interfaz gráfica de usuario. Algunos usuarios consideran muy sobria la interfaz gráfica de *ModusXXI*. Opinan que puede ser más atractiva si se le agregan gráficos y colores. Este aspecto puede ser considerado en desarrollos de versiones nuevas del programa.
- Mejorar la calidad visual de los gráficos en el archivo de ayuda. Esta observación ha sido ya atendida, y ha sido mejorada la resolución de los gráficos de dicho archivo. No obstante, no hay que dejar de considerar que la calidad visual también depende de la resolución de los monitores.

Nuevas funciones para versiones posteriores

- Generar intervalos y acordes. La generación de material armónico-polifónico sin duda es un aspecto de gran relevancia para el desarrollo de versiones posteriores.
- Generar melodías tonales. Al igual que el punto anterior, este aspecto es sin duda de gran importancia en el desarrollo de versiones nuevas.

- Posibilitar silencios en las melodías. Este aspecto es también muy relevante. Sin embargo, su implementación requiere, a nuestro juicio, una contextualización estilística; pues el desarrollo de un simple proceso aleatorio puede generar dificultades y perjuicios para la toma de dictados.
- Posibilitar la generación de melodías en compases compuestos. *ModusXXI* puede generar compases compuestos. 6/4 y 9/4 son compases compuestos. El compás compuesto, de uso común, que faltaría es el de 12/4. Su implementación debe ser considerada en versiones posteriores.

Conclusiones de la evaluación

ModusXXI ha sido evaluado con una aceptación y satisfacción por arriba del 80%, por parte de los usuarios encuestados. Si bien el número de usuarios encuestados es limitado, en el análisis de datos puede observarse una tendencia en la evaluación. El uso de *ModusXXI* para la enseñanza grupal, así como la representación de los conceptos musicales en la interfaz gráfica, son las debilidades encontradas por los usuarios en el programa. Por otro lado, los usuarios encuestados consideran mayoritariamente que *ModusXXI* puede mejorar su capacidad auditiva, que el programa es fácil de manejar y que para el estudio del *Modus Novus* y la música no tonal, *ModusXXI* puede ser una muy buena herramienta. Los comentarios de los usuarios señalan la escritura de las síncopas en *ModusXXI* como el error más considerable en la programación. No obstante, como ya ha sido señalado, éste es un problema de la biblioteca *jMusic* y está fuera de los alcances de la programación de *ModusXXI.*

ModusXXI ha sido tomado con gran entusiasmo por los usuarios encuestados. Si bien, el programa requiere desarrollos ulteriores, el programa cumple en lo general con el objetivo propuesto, que es la generación de material melódico no tonal al estilo del *Modus Novus*; de una manera simple y de fácil acceso.

5. Conclusiones

5.1 Logros

La aplicación de cómputo *ModusXXI* cumple con el objetivo planteado al principio de esta tesis, que es la generación de melodías, de acuerdo a la metodología propuesta por el *Modus Novus*, que sirvan para la realización de dictados o la generación de material de lectura y entonación durante el estudio de este libro. *ModusXXI* posee un método probabilístico de principios formales para la generación aleatoria de un número ilimitado de melodías diversas.

Desde el punto de vista de la calidad, señalada por Forouzan[1], *ModusXXI* cumple satisfactoriamente con los siguientes aspectos:

Operabilidad

- Precisión. *ModusXXI* cuenta con un método para atrapar errores y excepciones lo cual lo hace preciso en términos de estabilidad.
- Eficiencia. El 95% de los procesos de *ModusXXI* se realiza dentro de un segundo, por lo que se cumple con la condición de respuesta en tiempo real de ejecución.
- Seguridad. *ModusXXI* está programado en Java. Todas las operaciones se realizan dentro de la máquina virtual de Java. Esto evita el acceso directo del usuario y de *ModusXXI* al sistema, y consecuentemente, la generación de errores o fallos dentro del mismo.
- Respuesta oportuna. El procesamiento de datos en *ModusXXI* es realizado en un tiempo razonable.
- Facilidad de uso. La pruebas realizadas a *ModusXXI* muestran que la aplicación es fácil de usar y aprender.

Mantenimiento

- Capacidad de cambio y Flexibilidad. *ModusXXI* está desarrollado en el esquema POO (Programación Orientada a Objetos). Trabajar con *objetos* y *clases* facilita los cambios y nuevas implementaciones a la aplicación.

[1] Forouzan Behrouz A., *Foundations of Computer Science*, Thomson Books/Cole, 2003 Canada, páginas 202-205.

- Capacidad de prueba. Probar *ModusXXI* es muy fácil. Gracias a que está programado en Java puede ser ejecutado casi desde cualquier sistema operativo con un mínimo de requerimientos.

Capacidad de transferencia

- Reutilización del código. *ModusXXI* utiliza bibliotecas como *jMusic*. Asimismo las clases creadas especialmente para *ModusXXI* (`generadorRitmico`, `generadorMelodico`, etc.) pueden ser reutilizadas en nuevas versiones o aplicaciones.
- Interoperabilidad y portabilidad. *ModusXXI* es multiplataforma.

Documentación

- *ModusXXI* cuenta con un "Manual de Usuario" y un "Manual de Instalación". Ambos se encuentran en formato HTML y PDF, lo cual posibilita su acceso electrónico y en papel. Asimismo, la aplicación cuenta con un menú de ayuda el cual establece un vínculo al Manual de Usuario en formato HTML.

La interfaz gráfica de usuario, desarrollada también en el esquema POO, es un modelo de Interfaz Objeto-Acción (OAI) en un estilo de selección de menú; la cual es relativamente sencilla de aprender y requiere poco tiempo de entrenamiento, aún para usuarios inexpertos. *ModusXXI* cumple también satisfactoriamente con la Utilidad y Usabilidad señaladas por Shneiderman:[2]

- Utilidad y Usabilidad. La interfaz gráfica de *ModusXXI* facilita la generación de material melódico, su reproducción y visualización en forma de partitura. Su estructura a nivel *widget* (componentes de interfaz), permite al usuario, a través de la selección y definición de algunos parámetros, realizar los procesos anteriores de manera rápida, fácil y exitosa.

[2] Shneiderman Ben, *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (Trad. al Español, Jesús Sánchez Cuadrado), Pearson Educación, Madrid 2006, pags. 27-35.

ModusXXI cumple también con los siguientes principios establecidos por Shneiderman[3] respecto al desarrollo de interfaces de usuario:

- Nivel de competencia de los usuarios. *ModusXXI* puede ser utilizado tanto por usuarios expertos como novatos, sin detrimento de ninguno de los dos tipos de usuario.
- Identificación de las tareas. Las tareas a realizar por el programa están claramente establecidas. *ModusXXI* cuenta con sólo cuatro botones, cada uno de los cuales ejecuta una de las tareas de la aplicación: generar la melodía, reproducirla, detenerla y mostrar la partitura. Asimismo, sólo es posible acceder a los tres últimos si ha sido generada una melodía.
- Estilo de interacción. *ModusXXI* posee un estilo de interacción de selección de menús.
- Consistencia. La aplicación es consistente. La terminología y los procesos están estandarizados.
- Retroalimentación informativa y diálogos que conduzcan a la finalización de la tarea. Las ventanas de diálogo de *ModusXXI* ofrecen información sobre el estado del programa, así como de los posibles errores o fallos generados y su posible solución.
- Prevención de errores. La interfaz de usuario atrapa los posibles errores.
- Deshacer acciones de forma fácil. Cada vez que se desea hacer un cambio en la configuración de *ModusXXI*, una ventana de diálogo pregunta al usuario si está seguro que desea hacer dichos cambios.
- Soporte a la capacidad de control del usuario. El usuario posee un control absoluto de la interfaz de *ModusXXI*.
- Reducción de la carga de la memoria a corto plazo. *ModusXXI* utiliza terminología musical y no es necesaria una secuencia específica en la selección de parámetros. El usuario no necesita aprender códigos, nemotecnias o secuencias de acción complicadas.

[3] Ibidem páginas 76-94.

ModusXXI cumple satisfactoriamente con los requisitos de operabilidad, mantenimiento, capacidad de transferencia, además de contar con una documentación detallada para el usuario.

En la evaluación de uso y calidad obtuvo *ModusXXI* una calificación general por arriba del 80%; lo cual muestra la aceptación de la aplicación por parte de los usuarios encuestados.

5.2 Perspectivas de desarrollo ulterior

Si bien *ModusXXI* cumple con la mayoría de los requisitos de calidad planteados existen algunos aspectos que deben ser mejorados, reelaborados o ampliados. No obstante, llevar a cabo estas tareas se encuentra fuera de los objetivos y propósitos de la presente tesis (pues no afectan el funcionamiento general de la aplicación), por lo que únicamente los mencionaremos a continuación:

- Implementar una clase `InterfazGrafica`. Esto permitirá reducir el código de la clase que contiene la función `main`, logrando en consecuencia una implementación mejor diseñada dentro del esquema de orientación a objetos (POO).
- Escribir un manual del sistema. La información sobre el sistema se encuentra dentro del capítulo “Desarrollo”, de la presente tesis, no obstante habría que reescribirla dentro de un documento titulado “Manual del Sistema”.
- Implementar la aplicación como Applet. Esto permitirá correr *ModusXXI* desde Internet, sin necesidad de instalarlo en el disco duro de la computadora.
- Atender al principio de Universalidad. Si bien *ModusXXI* puede ser utilizado tanto por usuarios expertos como novatos, la actual versión está únicamente en idioma español; resultando necesario el desarrollo en un futuro de una versión en idioma inglés que facilite un acceso más universal.
- Corregir los problemas de escritura generados por la biblioteca *jMusic* en el archivo gráfico de partitura. Hay que incorporar las alteraciones en bemoles, así como representar las síncopas de manera correcta.

- Implementar procesos de transformación melódica. La implementación de estos procesos (inversión, retrogresión, transportación, etc.) permitirá a *ModusXXI*, abordar algunos de los aspectos de la metodología de adiestramiento auditivo de Friedmann.[4]
- Implementar secuencias polifónicas. Actualmente *ModusXXI* genera únicamente material musical monódico.
- Desarrollar una aplicación de cómputo generadora de secuencias melódico-polifónicas que atienda a los géneros, y sus estilos, tonales, modales y atonales.

[4] Friedmann Michael L., *Ear Training for the Twentieth-Century Music*, Yale University Press, USA 1990.

6. Bibliografía

Teoría Musical y Adiestramiento Auditivo

1. Edlund Lars, *Modus Novus*, AB Nordiska Musikförlaget/Edition Wilhelm Hansen Stockolm, 1963.
2. Edlund Lars, *Modus Vetus*, AB Nordiska Musikförlaget/Edition Wilhelm Hansen Stockolm, 1967.
3. Estrada Rodríguez Luis A., *Educación Musical Básica, Tomo I Entrenamiento Auditivo, Tomo II Nociones de Teoría y Notación Musicales, Armonía y Contrapunto*, Editorial Patria, México D.F., 1989.
4. Forte Allen, *The Strucuture of Atonal Music*, Yale University Press, USA 1973.
5. Friedmann Michael L., *Ear Training for the Twentieth-Century Music*, Yale University Press, USA 1990.
6. J∅rgen Jersild, *Laerebog i Rytmeläsning*, sin editorial, Copenhagen, 1962.
7. Mackamul Roland, *Lehrbuch der Gehörbildung Band I y II*, Bärenreiter-Verlag, Kassel, 1969.

Ciencias de la Computación y cómputo musical

1. Bell Doublas, *Java for Students*, Pearson Education Limited, Essex, 1999 (3ra. Ed., *Java para estudiantes*, tr. española de Alfonso Vidal Romero; Pearson Educación de México, México, 2003).
2. Boulanger Richard, *The Csound Book*, MIT Press, Cambridge Massachusetts, 2000.
3. Cope David, *Virtual Music*, MIT Press, Cambridge Massachusetts, 2001.
4. Flanagan David, *Java Examples in a Nutshell*, O´Reilly Media Inc., California, 1997.
5. Forouzan Behrouz A., *Foundations of Computer Science*, Thompson Learning Inc., California, 2003.
6. Reck Miranda Eduardo, *Composing Music with Computers*, Focal Press, Oxford Massachusetts, 2001.
7. Rowe Robert, *Machine Musicianship*, MIT Press, Cambridge Massachusetts, 2001.
8. Shneiderman Ben, *Designing the User interface*, Pearson Education, 2006 (4ta. Ed., *Diseño de interfaces de usuario*, tr. Española Jesús Sánchez Cuadrado; Pearson Educación, Murcia, 2006).

9. Stewart James, *Precalculus. Mathematics for Calculus*, Brooks/Cole Publishing Company, 2001 (3ra. Ed., *Precálculo*, tr. Española de Virgilio González Pozo; Internacional Thompson Editores, México, 2001).
10. Zhang Tony, *Teach Yourself C in 24 hours*, Macmillan Computer Publishing, Indianapolis, 2000 *(*2da. Ed. *Aprendiendo C en 24 Horas*, tr. española Sergio Kourchenko Barrena; México Prentice Hall, Pearson Educación de México, 2001).

Artículos y Tesis:

1. Cambouropulos Emilios, *Toward a General Computational Theory of Musical Structure*, The University of Edinburgh, Department of Artificial Intelligence, Ph.D. (Tesis Doctoral), Edinburgh 1998.
2. Dworak Paul E., «An Integrated Computer Curriculum for Music Ear Training», *Tecnological Directions in Music Lerning*, Ejournal.
3. Hess George Jr., «A Model for the Effective Use of Computer-assisted Instruction for Ear Training», en AAVV, *Tecnological Directions in Music Lerning*, Ejournal
4. Spangler Douglas, *Computer-assisted instruction in ear-training and its integration into undergraduate music programs during the 1998-99 academic year*, Universidad del Estado de Michigan (EE.UU.) 1999.

Páginas Web:

1. http://www.operone.de/ Catálogo de compositores. Actualización: 28/06/2007.
2. http://home.swipnet.se/sonoloco7/PhonoSuecia/edlund.html Revisiones de grabaciones. Actualización: 28/06/2007.
3. http://msu.edu/user/spangle9/thesis.html Tesis de Maestria. Actualización: 28/06/2007.
4. http://www.risingsoftware.com/auralia30/ Aplicación. Actualización: 28/06/2007.
5. http://www.earmaster.com Aplicación. Actualización: 28/06/2007.
6. http://www.algomusic.com/ Java Music Specification Language. 28/06/2007.
1. http://mitpress2.mit.edu/e-journals/Computer-Music-Journal/ Revista sobre Tecnología Musical. Actualización: 10/04/2007.
2. http://music.utsa.edu/tdml/ Tecnological Directions in Music Learning e-journal (Revista electrónica). Actualización: 28/06/2007.

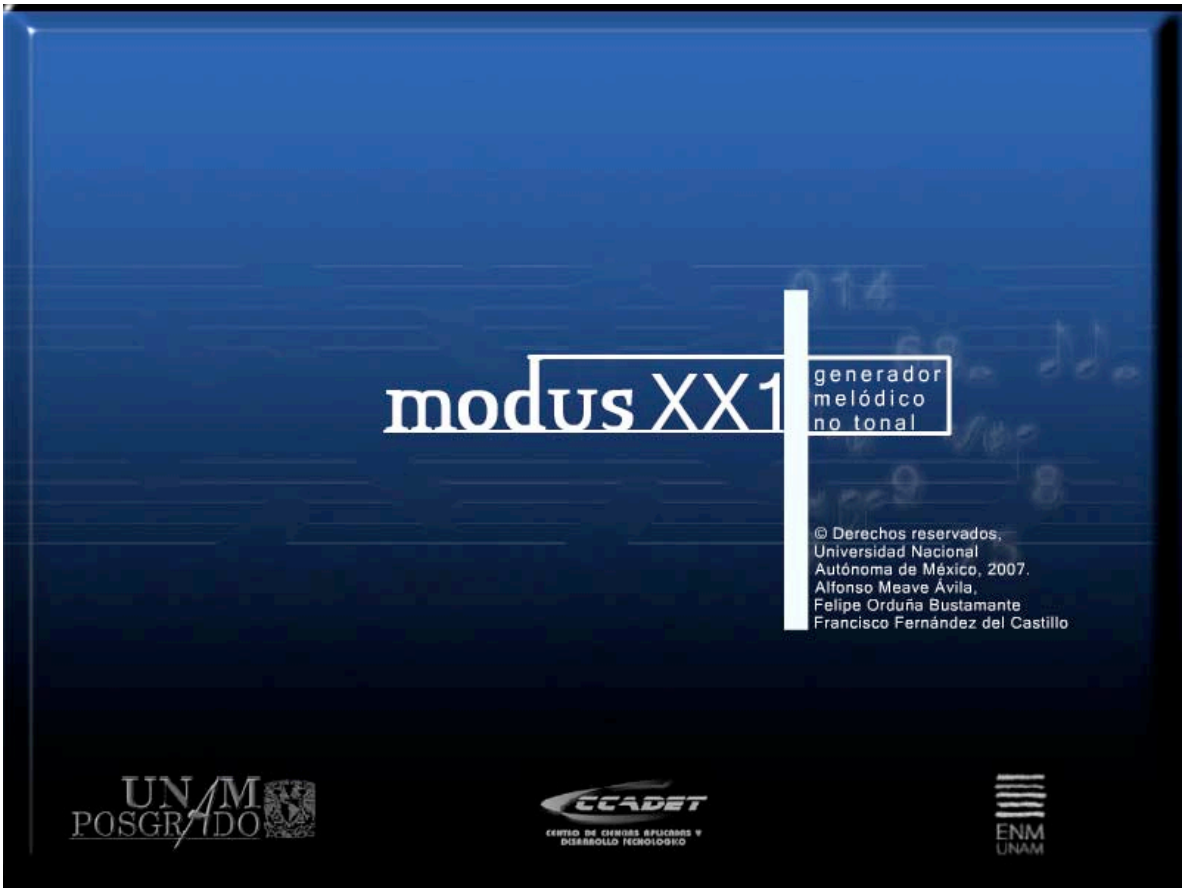
7. Apéndices

7. 1 Apéndice I. Tabla: Evaluación de Uso y Calidad de la aplicación *ModusXXI versión 1.0*

P1	P2	P3	P4	P5	P6	P7	P8	P9	Cuestionario
3	4	4	4	4	4	4	3	4	1
4	4	3	4	3	4	4	3	3	2
3	4	3	4	4	4	4	3	3	3
3	3	4	3	2	3	3	4	2	4
4	4	3	4	4	4	4	3	4	5
3	3	3	4	4	3	3	2	4	6
4	4	4	4	4	3	4	4	3	7
3	4	4	3	3	3	4	3	3	8
3	4	3	4	4	4	4	3	2	9
4	4	3	4	4	4	3	3	4	10
3	3	4	2	2	3	4	3	4	11
3	3	3	3	S/R	4	4	3	4	12
4	4	4	4	3	4	4	4	4	13
3	4	4	3	4	4	4	4	3	14
3	4	3	4	4	4	4	4	4	15
3	3	3	3	3	2	2	3	3	16
4	3	3	3	3	4	4	3	4	17
4	3	4	3	4	4	4	4	4	18
4	4	4	4	3	4	4	3	4	19
87,8	92,3	88,5	89,1	81,4	90,4	93,6	80,8	88,5	TOTAL
4	3	4	4	4	3	4	4	2	20
3	4	2	4	3	4	4	3	3	21
3	4	4	4	4	3	4	4	3	22
4	4	4	4	2	4	4	4	4	23
4	4	4	3	4	4	4	3	4	24
4	4	3	4	2	4	4	4	4	25
3	3	4	3	3	3	4	3	4	26
4	4	4	4	4	4	4	4	4	27
3	4	3	3	3	4	3	2	4	28
4	4	4	3	3	2	3	4	3	29
3	4	4	3	3	4	4	3	4	30
4	3	2	3	4	3	4	3	4	31
3	3	3	4	3	4	4	3	4	32
3	4	3	4	3	4	4	2	3	33
3	4	4	4	3	4	4	2	4	34
3	4	3	3	3	4	4	4	4	35
3	3	3	4	4	4	3	2	4	36
4	4	4	4	3	4	4	4	3	37
4	4	3	3	3	3	3	4	3	38
3	4	3	4	4	3	3	2	4	39

7.2 Apéndice II. Documentación

7.2.1 Manual de Instalación



Modus XXI Versión 1.0

Registro en trámite © 2007 Universidad Nacional Autónoma de México

Desarrollado por: Alfonso Meave Avila, Felipe Orduña Bustamante y Francisco Fernández del Castillo

Logo: Roque Alarcón

http://es.geocities.com/alfonsomeave Contacto: alfonsomeave@gmail.com

Derechos

ModusXXI es un proyecto de desarrollo tecnológico propiedad de la Universidad Nacional Autónoma de México. El usuario es responsable del manejo de *ModusXXI*. Ni el propietario ni los programadores se responsabilizan por daños que pudiera causar el manejo del programa.

Manual de instalación

¿Qué es *ModusXXI*?

ModusXXI es una aplicación de cómputo generadora de melodías no tonales, siguiendo la metodología *Modus Novus* de Lars Edlund[1]. *ModusXXI* está diseñado para realizar dictados melódicos no tonales y/o generar material melódico de entonación durante el estudio del *Modus Novus*. No obstante este diseño, *ModusXXI* puede usarse también para la generación de material melódico-musical que sirva para la composición.

¿Quién desarrolla *ModusXXI*?

ModusXXI es un proyecto de Tesis de Maestría en Tecnología Musical desarrollado dentro del programa de Posgrado de Maestría y Doctorado en Música de la Universidad Nacional Autónoma de México (UNAM) por el Lic. Alfonso Meave Avila bajo la tutoría del Dr. Felipe Orduña Bustamante y la asesoría del Mtro. Francisco Fernández del Castillo. Son instituciones participantes de este proyecto la Escuela Nacional de Música (ENM) y el Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET). *ModusXXI* utiliza la biblioteca *jMusic,* proyecto de investigación musical de la Universidad Tecnológica de Queensland, Australia desarrollado por Andrew Sorensen y Andrew Brown. http://jmusic.ci.qut.edu.au

¿En qué sistema operativo funciona *ModusXXI*?

ModusXXI está programado en lenguaje Java por lo que es multiplataforma. *ModusXXI* puede correr en sistemas operativos Microsoft Windows, Apple Machintosh, Linux, Solaris y todos aquellos con soporte para la Máquina Virtual de Java (jvm, por sus siglas en inglés).

[1] Edlund Lars, *Modus Novus*, Suecia, AB Nordiska Musikförlaget/Edition Wilhelm Hansen Stockolm, 1963.

Requerimientos del sistema:

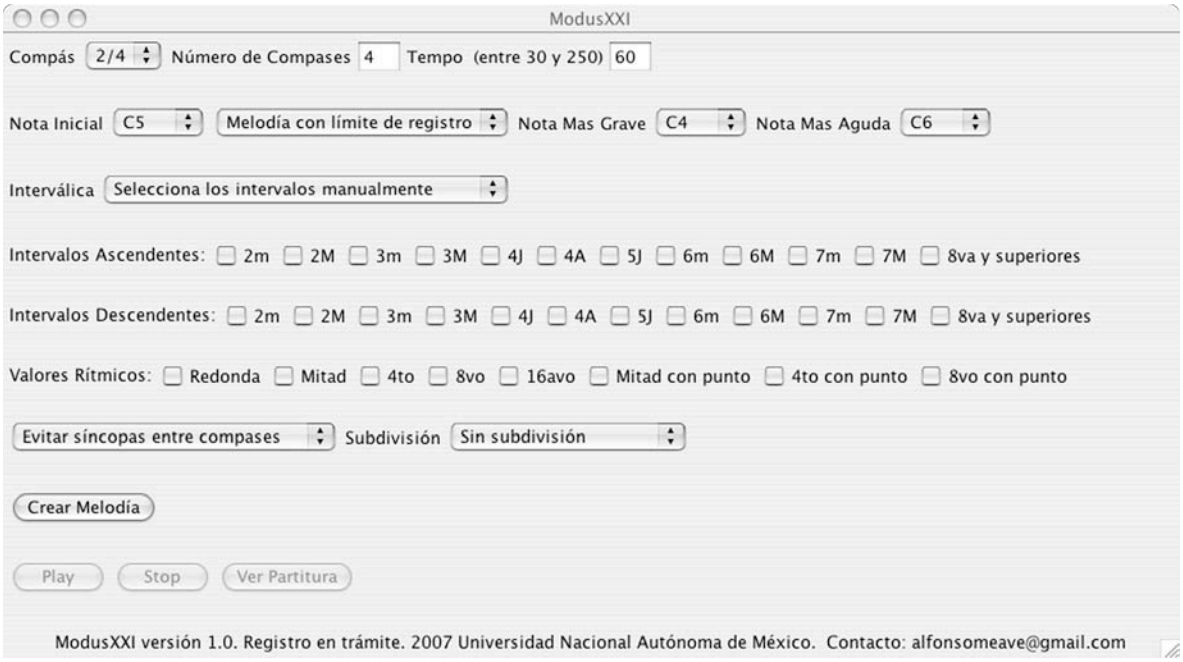
- Java Virtual Maschine (jvm) 1.4 o superior
- Sistema operativo Microsoft Windows, Mac OSX, Linux o Solaris.
- Tarjeta de audio con sintetizador interno de soporte General MIDI

Instalación

- Verifica si tienes instalada la Máquina Virtual de Java(jvm) 1.4 o superior. Si no la tienes instalada, puedes descargarla desde esta dirección electrónica: http://java.sun.com/j2se/1.4.2/ Instala el programa siguiendo las instrucciones del fabricante.
- Descarga el archivo *ModusXXI.zip* en la computadora.
- Descomprime el archivo *ModusXXI.zip* haciendo doble clic sobre él.
- Copia la carpeta *ModusXXI* en la carpeta de aplicaciones.
- Reinicia la computadora.

La primera melodía con *ModusXXI* en diez pasos

1. Haz doble clic en el archivo *ModusXXI.jar* (*ModusXXI* para usuarios Macintosh)
2. Aparecerá la siguiente ventana:



3. Haz clic en el selector de Compás y escoge el compás que desees. 2/4 está activado por default.
4. Introduce en el campo Duración en compases el número de compases que deseas dure la melodía (*ModusXXI* es capaz de generar melodías con una duración de hasta 67 cuartos o figuras equivalentes). Cuatro (4) compases aparecen por default.
5. Introduce en el campo Tempo la velocidad de la melodía a generar. Cuarto (negra) = 60 aparece por default.
6. Haz clic en el selector Interválica y selecciona Capítulo 1 (2m, 2M , 4J), o algún otro capítulo que desees.
7. Haz clic en algunas de las casillas de verificación del selector de Valores Rítmicos. Por ejemplo haz clic en 4to y 8vo. Una vez hecho clic, las casillas aparecerán activadas.
8. Presiona el botón "Crear Melodía".
9. Presiona el botón "Play". Deberás escuchar tu primera melodía.
10. Presiona el botón "Ver Partitura" para ver la melodía en notación musical.

¡Bienvenido al mundo *ModusXXI*!

IMPORTANTE. Si experimentas algún error o fallo:

Revisa que tengas instalada la Máquina Virtual de Java (jvm) versión 1.4 ó superior.

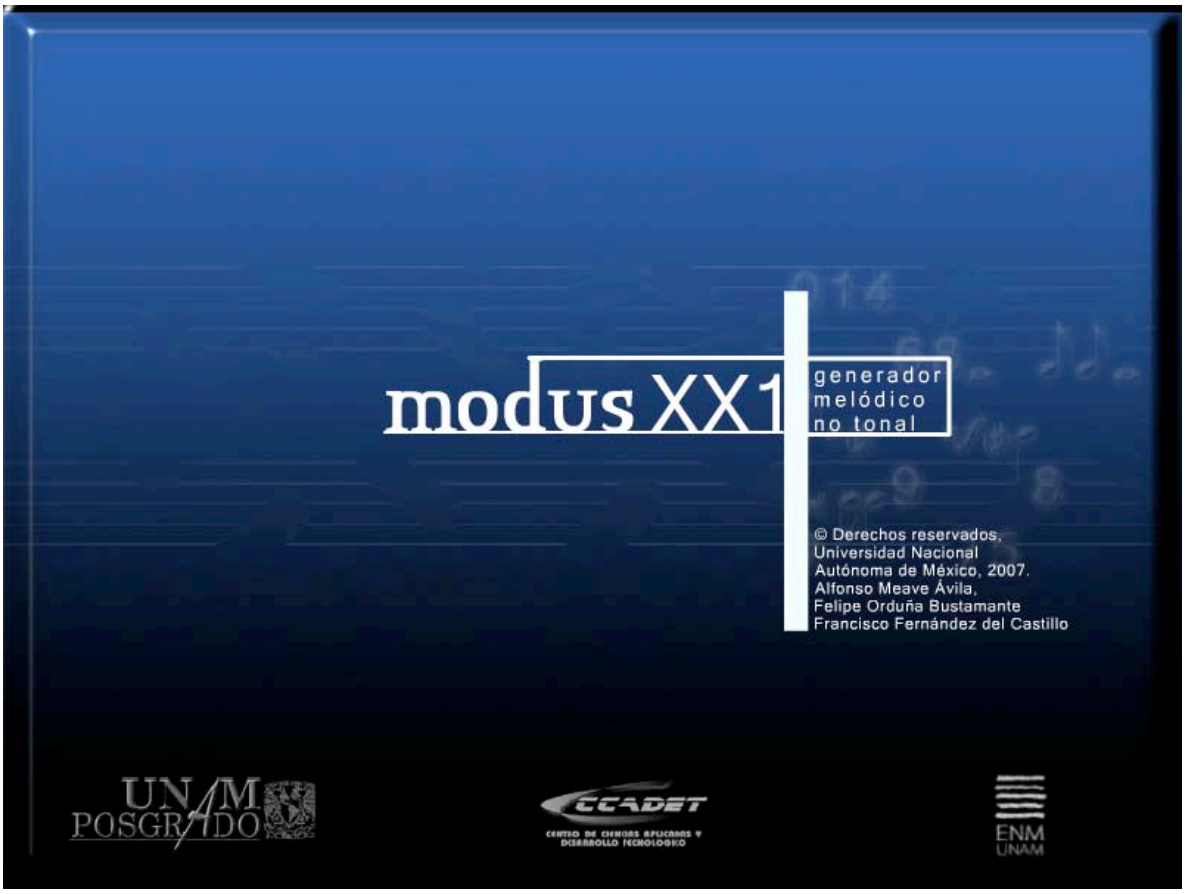
Si no la tienes instalada:

- Descárgala desde esta dirección electrónica: http://java.sun.com/j2se/1.4.2/
- Instala el programa siguiendo las instrucciones del fabricante
- Reinicia la computadora

Si tienes instalada alguna otra versión de java:

- Desinstala la versión que tengas instalada en tu computadora
- Descarga la versión 1.4 ó superior desde esta dirección electrónica: http://java.sun.com/j2se/1.4.2/
- Instala el programa siguiendo las instrucciones del fabricante
- Reinicia la computadora

7.2.2 Manual de Usuario



ModusXXI Versión 1.0

Registro en trámite © 2007 Universidad Nacional Autónoma de México

Desarrollado por: Alfonso Meave Avila, Felipe Orduña Bustamante y Francisco Fernández del Castillo

http://es.geocities.com/alfonsomeave Contacto: alfonsomeave@gmail.com

Derechos

ModusXXI es un proyecto de desarrollo tecnológico propiedad de la Universidad Nacional Autónoma de México. *ModusXXI* versión 1.0 puede generar errores en el sistema. El usuario es responsable del manejo de *ModusXXI*. Ni el propietario ni los programadores se responsabilizan por daños que pudiera causar el manejo del programa.

Manual de Usuario

INDICE

1. Introducción

ModusXXI es una aplicación de cómputo generadora de melodías no tonales, siguiendo la metodología *Modus Novus*[2] de Lars Edlund. *ModusXXI* está diseñado para realizar dictados melódicos no tonales y/o generar material melódico de entonación durante el estudio del *Modus Novus*. No obstante este diseño, *ModusXXI* puede usarse también para la generación de material melódico-musical que sirva para la composición. *ModusXXI* es un proyecto de Tesis de Maestría en Tecnología Musical desarrollado dentro del programa de Posgrado de Maestría y Doctorado en Música de la Universidad Nacional Autónoma de México (UNAM) por el Lic. Alfonso Meave Avila bajo la tutoría del Dr. Felipe Orduña Bustamante y la asesoría del Mtro. Francisco Fernández. Son instituciones participantes de este proyecto la Escuela Nacional de Música (ENM) y el Centro de Ciencias Aplicadas y Desarrollo Tecnológico (CCADET). *ModusXXI* utiliza la biblioteca *jMusic,* proyecto de investigación musical de la Universidad Tecnológica de Queensland, Australia desarrollado por Andrew Sorensen y Andrew Brown. (Para información respecto a la biblioteca *jMusic*, visite el siguiente url*:* http://jmusic.ci.qut.edu.au) *ModusXXI* está programado en lenguaje Java por lo que es multiplataforma. *ModusXXI* puede correr en sistemas operativos Microsoft Windows, Apple Machintosh, Linux, Solaris y todos aquellos con soporte para la Máquina Virtual de Java (jvm, por sus siglas en inglés).

[2] Edlund Lars, *Modus Novus*, Suecia, AB Nordiska Musikförlaget/Edition Wilhelm Hansen Stockolm, 1963.

2. Funcionamiento

ModusXXI es un generador aleatorio de material melódico no tonal. *ModusXXI* es capaz de crear una número ilimitado de melodías por medio de la definición, por parte del usuario, de los siguientes parámetros musicales:

- Compás
- Duración de la melodía en número de compases
- Tempo
- Interválica
- Nota inicial y registro (nota más grave y nota más aguda)
- Rítmica
- Subdivisión del compás (pulso)

Figura 1. Ventana principal de *ModusXXI*

2.1 Generación de material melódico

Para generar una melodía con *ModusXXI* es necesario realizar los siguientes pasos mínimos:

1. Haz clic en el selector de Compás y selecciona el compás que desees. 2/4 está activado por default.
2. Introduce en el campo Duración en compases el número de compases que deseas dure la melodía (*ModusXXI* es capaz de generar melodías con una duración de hasta 67 cuartos o figuras equivalentes). Cuatro (4) compases aparecen por default.
3. Introduce en el campo Tempo la velocidad de la melodía a generar. Cuarto (negra) = 60 aparece por default.
4. Haz clic en el selector Interválica y selecciona Capítulo 1 (2m, 2M , 4J) o algún otro capítulo u opción que desees practicar.
5. Haz clic en algunas de las casillas de verificación del selector de Valores Rítmicos. Por ejemplo haz clic en 4to y 8vo. Una vez hecho clic, las casillas aparecerán activadas.
6. Presiona el botón "Crear Melodía".
7. Presiona el botón "Play". Deberás escuchar la melodía generada.
8. Presiona el botón "Ver Partitura" para ver la melodía en notación musical.

2.2 Uso del *ModusXXI*

ModusXXI puede ser usado básicamente de dos maneras:

- Para generar material de dictado o entonación siguiendo la metodología *Modus Novus*.
- Para generar material melódico para la composición u otro fin.

IMPORTANTE. Si experimentas algún error o fallo:

Revisa que tengas instalada la Máquina Virtual de Java (jvm) versión 1.4 ó superior.

Si no la tienes instalada:

- Descárgala desde esta dirección electrónica: http://java.sun.com/j2se/1.4.2/
- Instala el programa siguiendo las instrucciones del fabricante
- Reinicia la computadora

Si tienes instalada alguna otra versión de java:

- Desinstala la versión que tengas instalada en tu computadora
- Descarga la versión 1.4 ó superior desde esta dirección electrónica: http://java.sun.com/j2se/1.4.2/
- Instala el programa siguiendo las instrucciones del fabricante
- Reinicia la computadora

2.2.1. Generación de material melódico siguiendo la metodología *Modus Novus*[3]

La metodología *Modus Novus* se caracteriza básicamente por llevar a cabo un estudio interválico no tonal de las relaciones melódico-armónicas entre los sonidos; y es precisamente la selección del material interválico en el *ModusXXI* la clave para generar melodías para dictados o entonación de acuerdo a la metodología *Modus Novus*. Para el principiante que desee iniciar el estudio del *Modus Novus* en coordinación con el *ModusXXI*, cabe hacer las siguientes recomendaciones que le facilitarán la toma de dictados:

- Selecciona al inicio compases simples como 2/4, 3/4 y 4/4.
- Selecciona duraciones menores o iguales a los cuatro compases.
- Selecciona tempos lentos. Cuarto = 60 es un buen inicio.
- Selecciona un registro de no más de dos octavas.
- Selecciona al inicio valores rítmicos isócronos, i.e. cuartos. Una vez dominados los dictados con ritmos isócronos puedes combinar distintas figuras rítmicas, i.e. cuartos y octavos.
- Deja activada la opción 

- Deja activada la opción 

- Activada la opción 

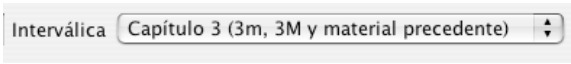
Activar esta opción va a tener como consecuencia que *ModusXXI* genera melodías con valores de cuarto o menores. Sin embargo, si sólo seleccionas valores de cuarto u octavo se van a generar grupos regulares de valores (los octavos siempre irán en pares), lo cual vuelve al dictado rítmicamente más simple y fácil de escribir.

[3] Ésta es sólo una breve introducción al uso del *ModusXXI* en coordinación con el *Modus Novus*. Una descripción concisa didáctico-metodológica sobre el uso simultáneo del *ModusXXI* y el *Modus Novus* se encuentra fuera de los objetivos de este manual.

2.2.1.1. El material interválico

ModusXXI posee en un selector o comboBox etiquetado como **Interválica.** Por default en este selector está activada la opción "**Selecciona los intervalos manualmente"**. Es necesario que esta opción esté activada si se desea seleccionar los intervalos desde las casillas de verificación. Si por el contrario, se desea hacer un estudio siguiendo la metodología *Modus Novus* es necesario hacer clic en el comboBox de **Interválica** y seleccionar alguna de las opciones de estudio que ofrece la opción **Modus Novus (Selecciona algún capítulo)**.[4]

Por ejemplo:



Si se selecciona esta opción *ModusXXI* generará melodías con los intervalos de estudio del Capítulo 3 del *Modus Novus*, es decir 2m, 2M, 3m, 3M, 4J y 5J. Lo mismo sucederá si se escoge cualquiera de las opciones intitulada **Capítulo…**

La opción **Sugerencia de estudio** es una propuesta metodológica de los desarrolladores de *ModusXXI*. Comprende trece opciones distintas de estudio las cuales sirven de estudio preparatorio para cada uno de los capítulos del *Modus Novus.* A diferencia de las opciones que ofrece la selección **Capítulo…**, **las sugerencias de estudio** generan melodías únicamente con el material interválico nuevo que presenta cada capítulo del *Modus Novus*, unido por intervalos de segunda mayor y menor. Por ejemplo, la opción

va a generar melodías únicamente con intervalos de 2m, 2M 3m y 3M. Esta opción elimina los materiales precedentes; en este caso la 4J y la 5J. Así, antes de generar melodías con la opción **Capítulo 3 (3m, 3M y material precedente)** se puede hacer un estudio previo usando la opción **3 (2m, 2M, 3m y 3M)**. (Para más información consulte la tabla **Grupos básicos de selección de material interválico en el *ModusXXI*** que se encuentra más abajo.)

Los capítulos 4, 8 y 11 del *Modus Novus* presentan ejemplos melódicos de la literatura musical. Por esta razón no son considerados dentro de las opciones que ofrece 

.

Las opciones 4, 8 y 11 de 

ofrecen síntesis de los materiales precedentes. La opción 13 permite la generación de material melódico con intervalos iguales o superiores a la octava. (Para más información consulte la tabla Grupos básicos de selección de material interválico en el *ModusXXI* que se encuentra más abajo.)

2.2.2. Generación de material melódico de manera libre

Para generar material interválico de manera libre sólo es necesario seleccionar la opción **Selecciona los intervalos manualmente** y activar manualmente las casillas de verificación de los intervalos:

Esta opción permite asimismo la selección de la dirección (ascendente o descendente) de los intervalos.

Todas las demás opciones pueden ser utilizadas de la misma manera que en la metodología *Modus Novus*.

3. Parámetros de selección del *ModusXXI*

1. Selector de compás. Activa el compás en que se desea generar la melodía. Las opciones van de 2/4, 3/4, 4/4,…hasta 9/4.
2. Selector de duración de la melodía. Aquí debe introducirse en números enteros el número de compases que se desea que dure la melodía. Por ejemplo, si se desea que dure la melodía dos compases se escribe 2, si se desea que dure cinco se escribe 5, etc. *ModusXXI* es capaz de generar melodías con una duración de hasta 67 cuartos o figuras equivalentes.
3. Selector de tempo. Aquí se debe introducir un valor numérico entero que defina la velocidad de la melodía que va a generar *ModusXXI*. El valor numérico del tempo debe encontrarse entre 30 y 250 (pulsos por segundo).
4. Selectores de registro. Los selectores de registro comprenden cuatro ComboBox:
    a) Nota Inicial. Escoge la nota que se desea al inicio de la melodía a generar por *ModusXXI.* La selección va de Do0 a Sol10 (Registro que

permite el protocolo MIDI, donde Do0 equivale a 0 y Sol10 a 127). Do5 está activado por default.

b) Melodía con límite de registro o Melodía sin límite de registro. Melodía sin límite de registro desactiva la nota más grave y la nota más aguda, permitiendo que los límites de registros sean Do0 y Sol10. Melodía con límite de registro activa la nota más grave y la nota más aguda.

c) Nota Más Grave. Selecciona la nota límite en el registro grave para la generación de la melodía. La selección va de Do0 a Sol10.

d) Nota Más Aguda. Selecciona la nota límite en el registro agudo para la generación de la melodía. La selección va de Do0 a Sol10.

5. Selector de material interválico de acuerdo a la metodología *Modus Novus*. Este selector posee tres grupos básicos de activación del material interválico:

a) "Selecciona los intervalos manualmente". Al escoger esta opción del selector, el usuario debe activar manualmente las casillas de verificación de los "Selectores manuales de intervalos". Esta opción ofrece también la selección de la dirección, ascendente o descendente, del intervalo. Por ejemplo, se puede seleccionar únicamente el intervalo de 5J (quinta justa) ascendente. Si se desea que el intervalo de 5J se presente tanto ascendentemente como descendentemente es necesario activar ambas casillas de verificación. ¡Para hacer una selección manual de los intervalos es necesario activar esta opción!

b) "Modus Novus". Esta opción requiere la selección específica de algún capítulo del *Modus Novus*. Al seleccionar algún capítulo y pulsar el botón Crear Melodía se activarán automáticamente las casillas de verificación de los intervalos, ascendentes y descendentes, correspondientes al capítulo del *Modus Novus* que se desee practicar. Por ejemplo, al seleccionar Capítulo 1 los intervalos de segunda menor, segunda mayor

y cuarta justa (ascendentes y descendentes) serán activados automáticamente.

c) **"Sugerencia de estudio"**. Esta opción sirve de estudio preparatorio para los capítulos del *Modus Novus*. A diferencia del material generado con las opciones de "**Modus Novus",** las opciones de **"Sugerencia de estudio"** crean melodías únicamente con el material interválico nuevo que contiene cada capítulo del *Modus Novus*; unido a través de intervalos de segunda menor y mayor. Por ejemplo en **3 (2m, 2M, 3m, 3M)** el material nuevo de estudio son los intervalos de tercera menor y tercera mayor. Así, las melodías son generadas por *ModusXXI* utilizando únicamente dicho material nuevo (3m y 3M en este caso) unido por intervalos de segunda menor y mayor. La diferencia entre seleccionar **Capítulo 3** y **3 (2m, 2M, 3m, 3M)** es que al seleccionar **Capítulo 3** también van a ser seleccionados los intervalos de cuarta justa, quinta justa, así como los de segunda menor y segunda mayor; mientras que al seleccionar **3 (2m, 2M, 3m, 3M)** únicamente será seleccionado el material interválico nuevo del capítulo 3 del *Modus Novus* junto con intervalos de segunda mayor y menor. Esta opción permite un estudio más detallado del material nuevo que presenta cada capítulo del *Modus Novus.*

6. **Selectores manuales de intervalos.** Son casillas de verificación que sirven para seleccionar manualmente los intervalos en dirección tanto ascendente como descendente. Los intervalos a seleccionar van de la segunda menor a la séptima mayor. Existe además una casilla de verificación (**8va y superiores)**, ascendente y descendente, que reúne un grupo de intervalos iguales o mayores a la octava. El registro límite de dichos intervalos es de tres octavas. Recuerda que **para hacer una selección manual de los intervalos es necesario activar la "selección manual de intervalos" en el Selector de material Interválico.**

7. **Selectores de valores rítmicos.** Activan los valores rítmicos que se desea que tenga la melodía. Dichos valores son redonda, mitad, cuarto, octavo, dieciseisavo, mitad con punto, cuarto con punto y octavo con punto.
8. **Selector de melodía sin síncopas entre compases ó Melodía con síncopas entre compases.** La generación de figuras rítmicas en *ModusXXI* es también un proceso aleatorio, el cual tiene como límites de generación y selección de las figuras rítmicas a:
    a) **La duración total en compases de la melodía.** *ModusXXI* escoge valores rítmicos de forma aleatoria hasta llenar el espacio total de compases definidos por el usuario. Si el usuario selecciona **Melodía con síncopas entre compases**, *ModusXXI* va a hacer su selección de valores rítmicos sin tomar en cuenta a la unidad de compás. La consecuencia de esto va a ser la generación de material melódico posiblemente con síncopas entre compases. La selección de **Melodía con síncopas entre compases** va a desactivar el **selector de unidad de subdivisión**.
    b) **La duración en cuartos de cada compás (Unidad de medida).** Al activar **Melodía sin síncopas entre compases** *ModusXXI* selecciona las figuras rítmicas tomando en cuenta tanto la duración total en compases como la unidad de medida del compás; i.e. dos para 2/4, tres para 3/4, cuatro para 4/4, etc. Como resultado las síncopas entre compases son eliminadas y consecuentemente las melodías tendrán siempre compases téticos. Por default está activada la función **Melodía sin síncopas entre compases.**
    c) **El número de pulsos en cuartos o octavos de cada compás (unidad de subdivisión).** Al seleccionar el material rítmico *ModusXXI* también puede tomar en cuenta, aparte del total de compases y la unidad de medida, la unidad de tiempo o pulso. La consecuencia va a ser no sólo la eliminación de síncopas dentro del compás, sino también la imposibilidad de seleccionar valores superiores a la unidad de subdivisión

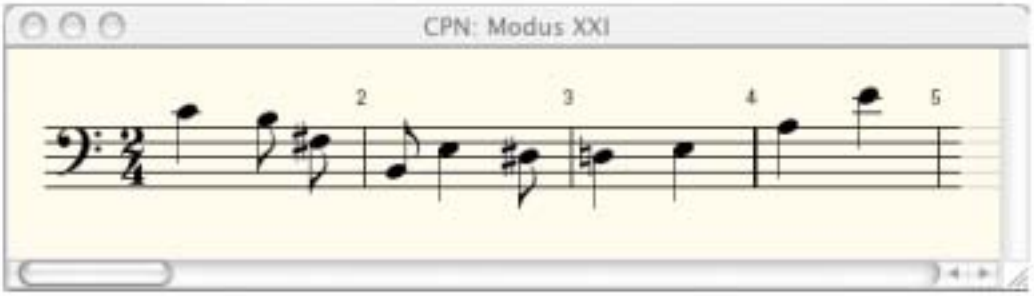
seleccionada. Esta función va a depender del **selector de unidad de subdivisión.**

9. **Selector de unidad de subdivisión.** Este selector ofrece tres posibles activaciones:
    a) **Sin subdivisión.** Activada por default, toma como parámetro de selección del material rítmico a la unidad de compás. Consecuencia de su activación es la generación de material rítmico tético pero con posibilidad de síncopas dentro del compás.
    b) **Subdivisión en cuartos.** Toma como parámetro de selección del material rítmico al cuarto. La activación de esta opción genera secuencias de material rítmico de cuartos, octavos y dieciseisavos. Consecuentemente, se imposibilita la selección de valores rítmicos superiores al cuarto.
    c) **Subdivisión en octavos.** Toma como parámetro de selección del material rítmico al octavo. La activación de esta opción genera secuencias de material rítmico de octavos y dieciseisavos. Consecuentemente, se imposibilita la selección de valores rítmicos superiores al octavo.
10. **Botón generador de la melodía ("Crear Melodía").** Cada vez que se oprima este botón se va a generar una nueva melodía.
11. **Botón para mostrar la partitura ("Ver partitura").** Al oprimir este botón va a aparecer en la pantalla una nueva ventana con la notación musical de la melodía que se generó.
12. **Botón de reproducción ("Play").** Al oprimir este botón va a reproducirse la melodía generada.
13. **Botón para detener la reproducción ("Stop").** Al oprimir este botón se va a detener la reproducción de la melodía.

4. Algunas funciones que ofrece la ventana que contiene la partitura

La generación del archivo gráfico de la partitura, así como la reproducción vía MIDI es llevada a cabo en *ModusXXI* por medio de la biblioteca *jMusic*. La biblioteca *jMusic* es un proyecto de investigación musical de la universidad tecnológica de Queensland, Australia. La biblioteca *jMusic* ofrece un conjunto de herramientas para la composición y el procesamiento de audio en lenguaje Java.

Al oprimir el botón Ver partitura se abrirá una nueva ventana como ésta:



Esta ventana posee cuatro opciones de menú proporcionadas por la biblioteca *jMusic*: File, Tools, Play y View. A continuación nos limitaremos a describir las funciones más relevantes de estos menús para el usuario del *ModusXXI*.

- Guardar la melodía como archivo MIDI: selecciona *File* y después selecciona *Save as MIDI File*.
- Cerrar la ventana: selecciona *File* y después selecciona *close*.
- Cambiar instrumento GeneralMidi (*ModusXXI* genera por default las melodías con el *patch* GrandPiano de GeneralMidi), volumen o tempo: selecciona *Tools* y después selecciona *Set Parameters*. Aparecerá una nueva ventana, allí hay que seleccionar el parámetro a modificar, hacer clic en *Apply* y después en *Close*.
- Reproducir desde la ventana de la partitura: selecciona *Play* y después selecciona *Play All*.
- Detener la reproducción: selecciona *Play* y después selecciona *Stop Playback*.

Si se desea más información respecto a la biblioteca *jMusic* y su manejo se puede remitir al siguiente url: http://jmusic.ci.qut.edu.au

¡Que tengas mucho éxito y diversión con *ModusXXI*!

Contacto o comentarios:

Url: http://es.geocities.com/alfonsomeave

e-mail: alfonsomeave@gmail.com

7.3 Apéndice III. Ejemplos de código fuente

A continuación presentamos el código fuente de las clases más importantes de *ModusXXI*.

7.3.1 GeneradorMelodico.java

```
public class GeneradorMelodico {

    private static int [] valoresIntervalicos;

    /** Creates a new instance of GeneradorMelodico */

    public GeneradorMelodico(int [] intervalos) {
        valoresIntervalicos = intervalos;
    }

    public static int [] generarNotas(int notaInicial, int n){
      return generarNotasConRegistro(notaInicial, 0, 127, n);
    }


    public static int [] generarNotasConRegistro(int notaInicial,
            int notaMasGrave, int notaMasAguda, int numeroDeNotas){

        int arrayNotas [] = new int [numeroDeNotas];

        arrayNotas [0] = notaInicial;
        for ( int i = 1 ; i < numeroDeNotas ; i++) {


            int iteraciones = 0;
            int indiceAleatorio;
            int nuevaNota;

            do  {

                if (iteraciones++ > 1000) {
                    return null;
                }
                indiceAleatorio = (int) Math.floor(Math.random() *
                    valoresIntervalicos.length);
                nuevaNota = arrayNotas[i - 1]  + valoresIntervalicos
                    [indiceAleatorio];
            }  while (nuevaNota > notaMasAguda || nuevaNota < notaMasGrave );

            arrayNotas [i] = nuevaNota;
        }
        return arrayNotas;

    }
}
```

7.3.2 GeneradorRitmico.java

```
public class GeneradorRitmico {

       private static double [] valoresRitmicos;

       public GeneradorRitmico(double [] valores) {
         valoresRitmicos = valores;
    }

       public static double [] generarRitmosPorCompas(double duracionTotal){
         int valorAleatorio = 0;
         double duracion = 0;

         DoubleList temp = new DoubleList();
         for (int i = 0 ; duracion < duracionTotal ; i++) {

                   valorAleatorio = (int) Math.floor(Math.random() *
                           valoresRitmicos.length);
             temp.add(valoresRitmicos[valorAleatorio]);
             duracion = duracion + valoresRitmicos [valorAleatorio];
         }

         return temp.toArray();
       }

       public static double [] generarRitmosXCompasNoSincopado(double
       duracionTotal, double compas){
         int valorAleatorio = 0;
         double duracion = 0;
         double nuevaFiguraRitmica = 0;
         double pulso = 0;

         DoubleList temp = new DoubleList();
         for (int i = 0 ; duracion < duracionTotal ; i++) {

             int iteraciones = 0;

              do {

                 if (iteraciones++ > 1000) {
                      return null;
                 }

              valorAleatorio = (int) Math.floor(Math.random() *
              valoresRitmicos.length);
                 nuevaFiguraRitmica = valoresRitmicos[valorAleatorio];
             } while ( pulso + nuevaFiguraRitmica > compas );

             temp.add(nuevaFiguraRitmica);
             duracion = duracion + nuevaFiguraRitmica;
             pulso = pulso + nuevaFiguraRitmica;
             if ( pulso >= compas )
                 pulso = pulso - compas;
         }

         return temp.toArray();
    }
}
```

7.3.3 Ejercicio.java

```
public class Ejercicio implements JMC, ConstantesMusicales {
    private static double [] secuenciaRitmica;
    private static int [] secuenciaMelodica;
    private static int numerador = 4;
    private static int denominador = 4;
    public static double pulso;

    public Ejercicio(double [] ritmos, int [] notas) {
        secuenciaRitmica = ritmos;
        secuenciaMelodica = notas;
    }
    public static int length() {
        return secuenciaRitmica.length;
    }
    public static double ritmo(int n) {
        return secuenciaRitmica[n];
    }
    public static int nota(int n) {
        return secuenciaMelodica[n];
    }
    public static double [] ritmos() {
        return secuenciaRitmica;
    }
    public static int [] notas() {
        return secuenciaMelodica;
    }
    public static void asignarNumerador(int num) {
        numerador = num;
    }
    public static void asignarDenominador(int den) {
        denominador = den;
    }
    public void playJM(int tempo) {
       if (secuenciaRitmica == null || secuenciaMelodica == null) {
            return;
        }
        Note [] jm_notas = new Note[length()];
        for ( int i = 0 ; i < length() ; i++){
            jm_notas[i] = new Note(nota(i), ritmo(i));
        }
        double [] acentos = {0.0}; //para denominador = cuarto
        double numeroDeCuartosPorCompas = 4.0 * (double) numerador / denominador;
        Phrase jm_frase = new Phrase(jm_notas);
        Part jm_parte = new Part(jm_frase);
        Score jm_score = new Score(jm_parte);
        Mod.accents(jm_score, numeroDeCuartosPorCompas,
                acentos, Note.MAX_DYNAMIC); //numeroDeCuartosPorCompas

        jm_score.setTimeSignature(numerador, denominador);
        jm_score.setVolume(50);
        jm_score.setTitle("Modus XXI");
        jm_score.setTempo(tempo);
        Play.midi(jm_score, false);
    }
    public void stopJM(int tempo) {
       Play.stopMidi();
    }
    public void viewJM(int tempo) {
        new Notate(jm_score, 100, 100);
    }
}
```

7.3.4 GeneradorDeEjercicios.java

```
public class GeneradorDeEjercicios implements JMC {
    private static GeneradorRitmico gr;
    private static GeneradorMelodico gm;
    public GeneradorDeEjercicios(double [] valoresRitmicos, int [] notas) {
        gr = new GeneradorRitmico (valoresRitmicos);
        gm = new GeneradorMelodico (notas);
    }
    public static Ejercicio generarEjercicio ( int numerador, int denominador,
             int numeroDeCompases, double pulso, int notaInicial, int
                    notaMasGrave, int notaMasAguda) {
        double compas = QUARTER_NOTE * 4.0 * (double) numerador / denominador;
        double subdivision = QUARTER_NOTE * 4.0 * pulso;
        double duracionTotal =  numeroDeCompases * compas;

        int [] notas;
        double [] ritmos = gr.generarRitmosXCompasNoSincopado (duracionTotal,
                           subdivision);
        if (ritmos == null) {
            notas = null;
        } else {
            notas = gm.generarNotasConRegistro(notaInicial, notaMasGrave,
                    notaMasAguda,
                ritmos.length);
      }
      Ejercicio ejercicio = new Ejercicio (ritmos, notas);

        ejercicio.asignarNumerador (numerador);
        ejercicio.asignarDenominador (denominador);

        return ejercicio;
    }

    public static Ejercicio generarEjercicio ( int numerador, int denominador,
               int numeroDeCompases, double pulso, int notaInicial) {

        double compas = QUARTER_NOTE * 4.0 * (double) numerador / denominador;
        double subdivision = QUARTER_NOTE * 4.0 * pulso;
        double duracionTotal =  numeroDeCompases * compas;

        int [] notas;
        double [] ritmos = gr.generarRitmosXCompasNoSincopado (duracionTotal,
                    subdivision);
        if (ritmos == null) {
            notas = null;
        } else {
         notas = gm.generarNotas(notaInicial, ritmos.length);
      }
        Ejercicio ejercicio = new Ejercicio (ritmos, notas);

        ejercicio.asignarNumerador (numerador);
        ejercicio.asignarDenominador (denominador);
        return ejercicio;
    }

public static Ejercicio generarEjercicio ( int numerador, int denominador,
               int numeroDeCompases, int notaInicial, int notaMasGrave, int
                    notaMasAguda) {

        double compas = QUARTER_NOTE * 4.0 * (double) numerador / denominador;
        double duracionTotal =  numeroDeCompases * compas;

        int [] notas;
```

```
        double [] ritmos = gr.generarRitmosPorCompas (duracionTotal);
        if (ritmos == null) {
            notas = null;
        } else {
            notas = gm.generarNotasConRegistro(notaInicial, notaMasGrave,
                    notaMasAguda,
                ritmos.length);
       }

        Ejercicio ejercicio = new Ejercicio (ritmos, notas);
        ejercicio.asignarNumerador (numerador);
        ejercicio.asignarDenominador (denominador);
        return ejercicio;
    }

    public static Ejercicio generarEjercicio ( int numerador, int denominador,
               int numeroDeCompases, int notaInicial) {
        double compas = QUARTER_NOTE * 4.0 * (double) numerador / denominador;
        double duracionTotal =  numeroDeCompases * compas;

        int [] notas;
        double [] ritmos = gr.generarRitmosPorCompas (duracionTotal);
        if (ritmos == null) {
            notas = null;
        } else {
            notas = gm.generarNotas(notaInicial, ritmos.length);
        //if (notas == null) {
            //return null;
        }
        Ejercicio ejercicio = new Ejercicio (ritmos, notas);

        ejercicio.asignarNumerador (numerador);
        ejercicio.asignarDenominador (denominador);
        return ejercicio;
    }
}
```

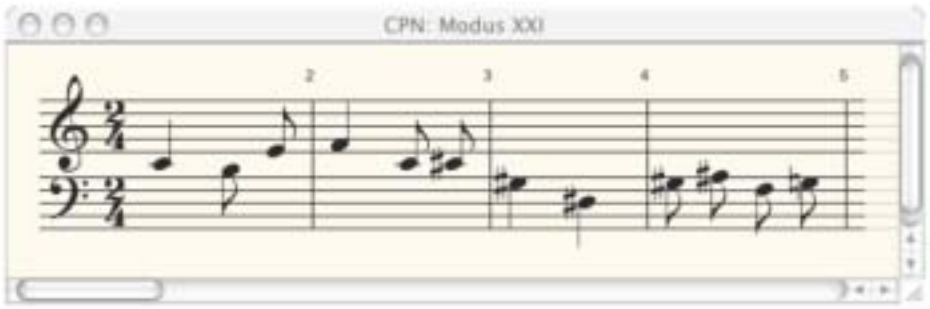
7.3.5 ModusXXI.java

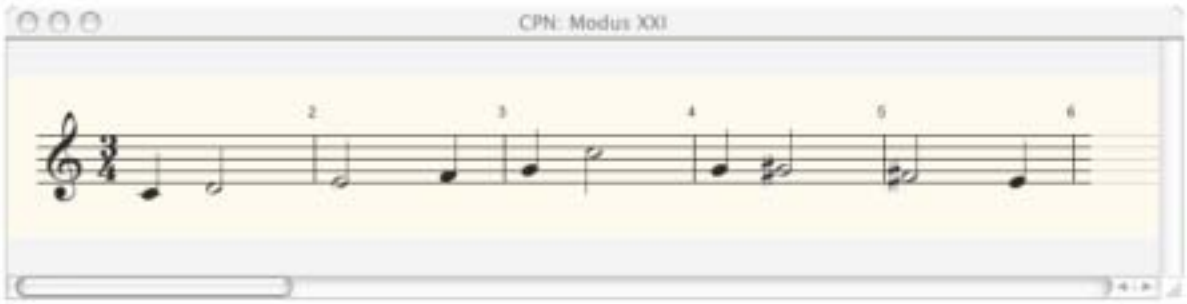
El código completo de esta clase es de 49 páginas. Presentamos únicamente su esquema básico.

```
public class ModusXXI extends JFrame implements JMC, ActionListener,
ConstantesMusicales {

    public ModusXXI() {
        super("ModusXXI");
        initComponents();
        setVisible(true);
    }

    public void initComponents() {
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

    public void actionPerformed(ActionEvent evt){
    }


   GeneradorDeEjercicios gen = new GeneradorDeEjercicios(valoresRitmicos,
                    valoresIntervalicos);

    }

    public static void main(String[] args ) {
        new ModusXXI();
    }

    private void abrirAbout() {
        aboutBox.setVisible(true);
    }

    private void abrirAyuda() {


    private void abrirWebSite() {

    }

    public int verificarCrearMelodia() {

        }

    }
```

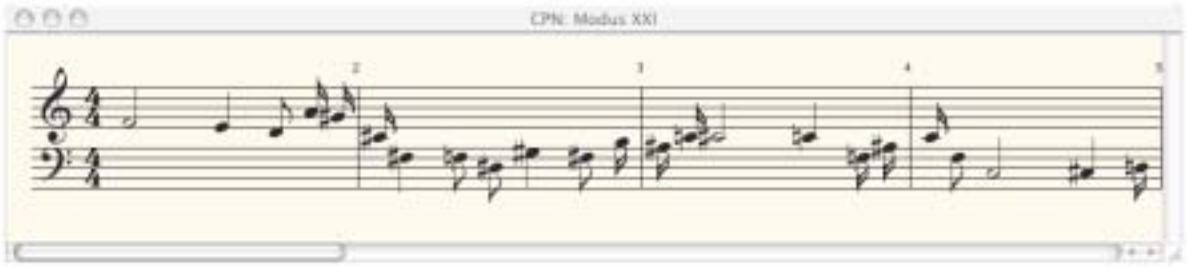
7.4 Apéndice IV. Ejemplos de melodías generadas con *ModusXXI*

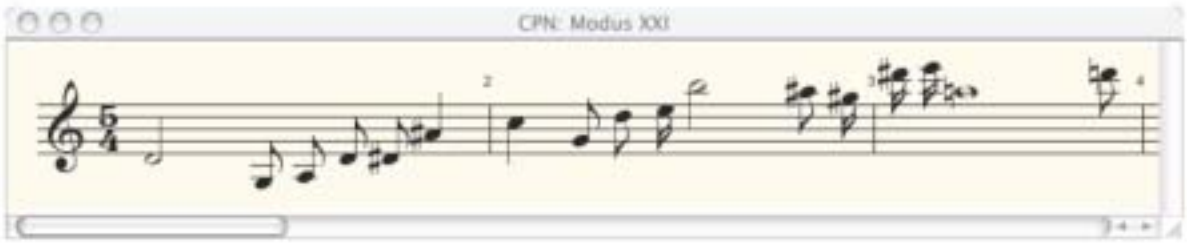
Melodías basadas en el capítulo 1 del *Modus Novus*



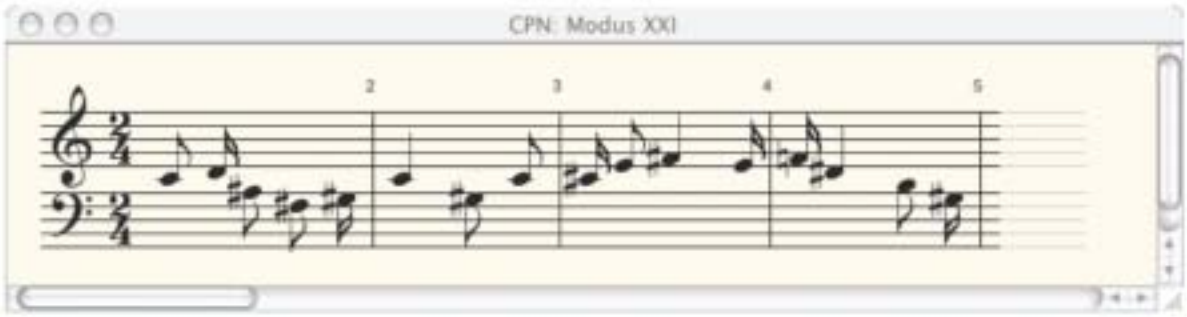


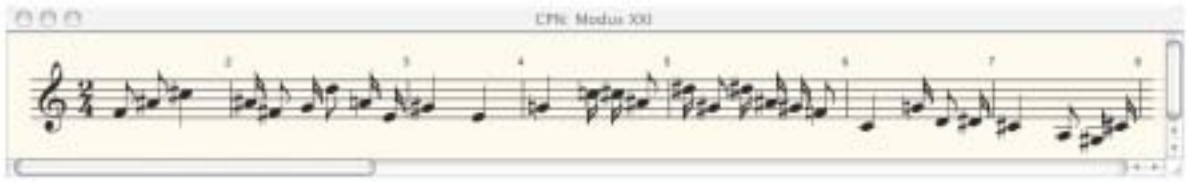
Melodías basadas en el capítulo 2 del *Modus Novus*



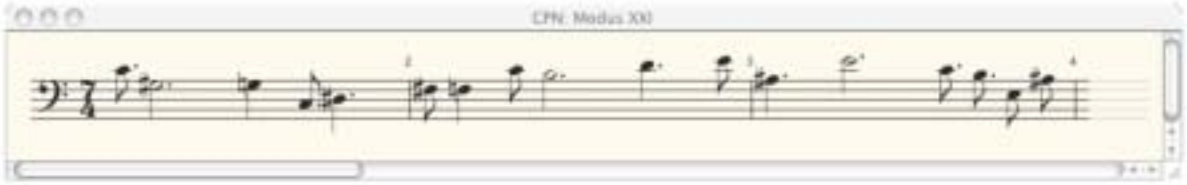


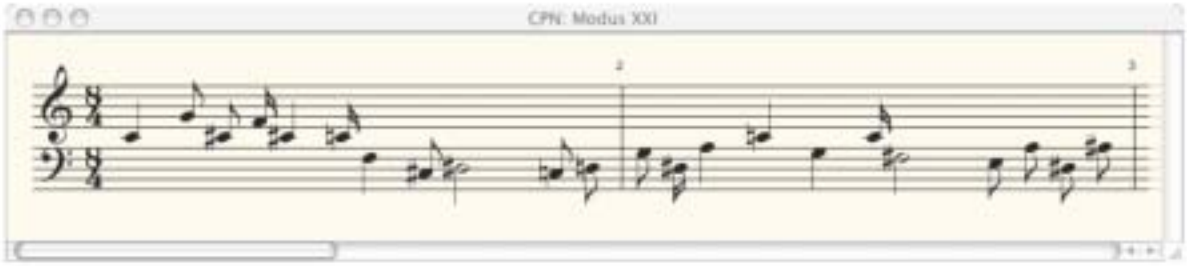
Melodías basadas en el capítulo 3 del *Modus Novus*



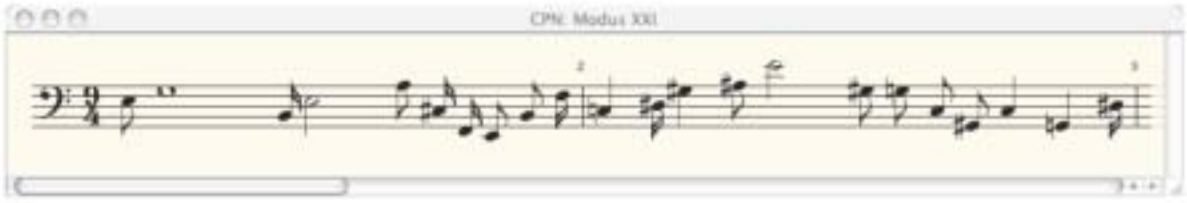


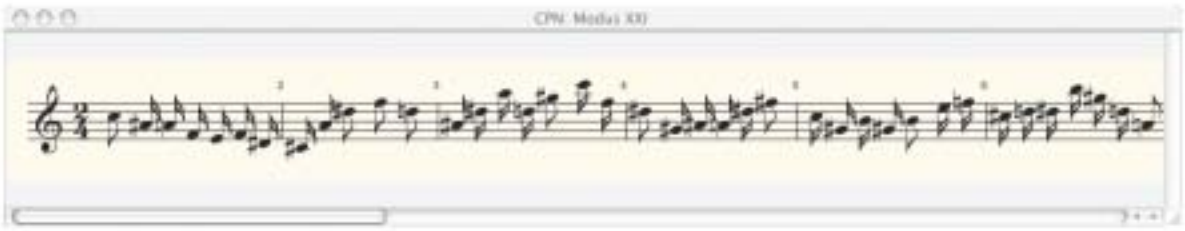
Melodías basadas en el capítulo 5 del *Modus Novus*



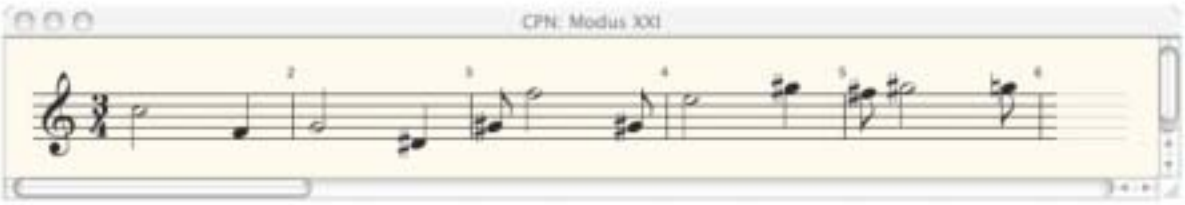


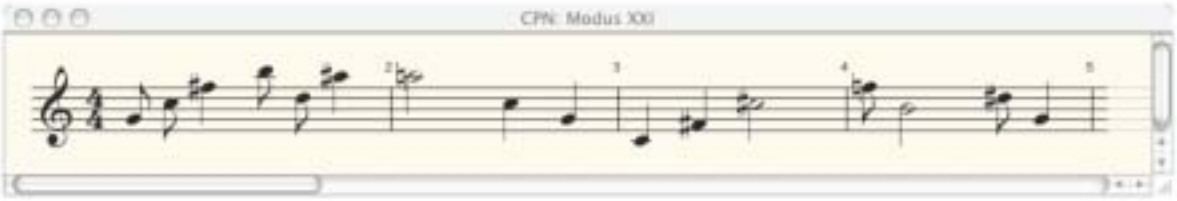
Melodías basadas en el capítulo 6 del *Modus Novus*



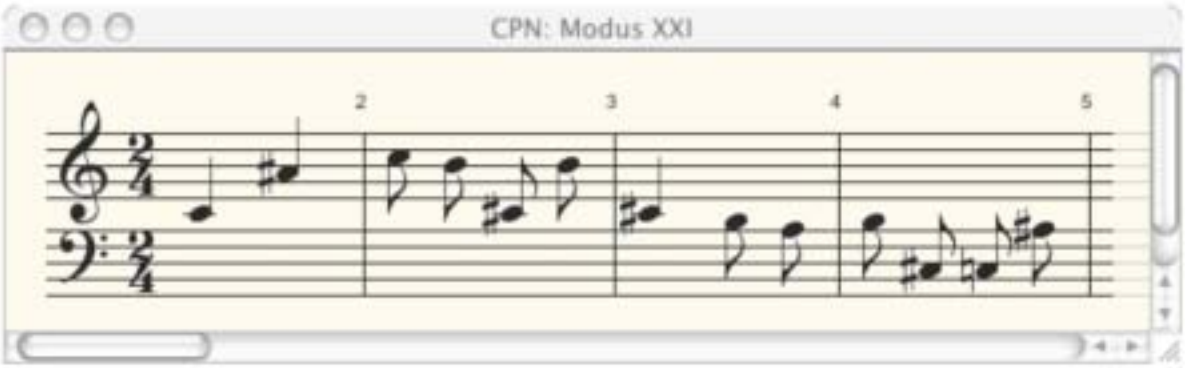


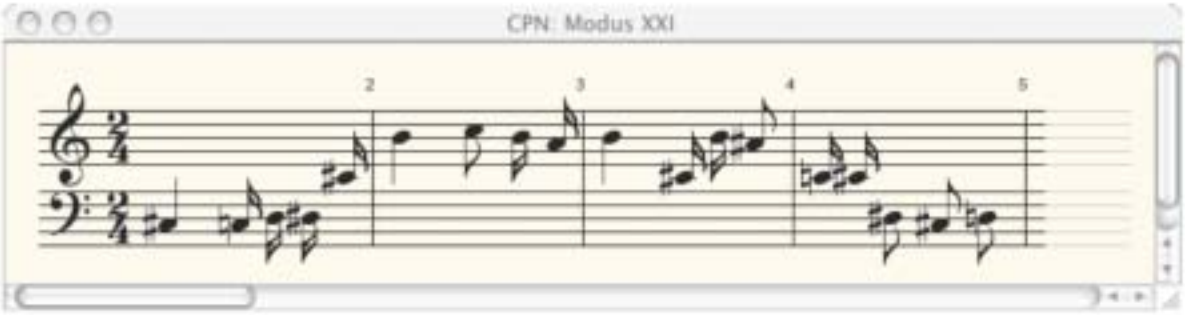
Melodías basadas en el capítulo 7 del *Modus Novus*



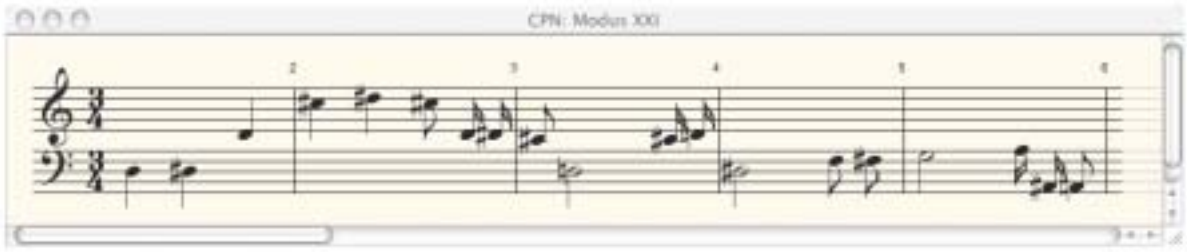


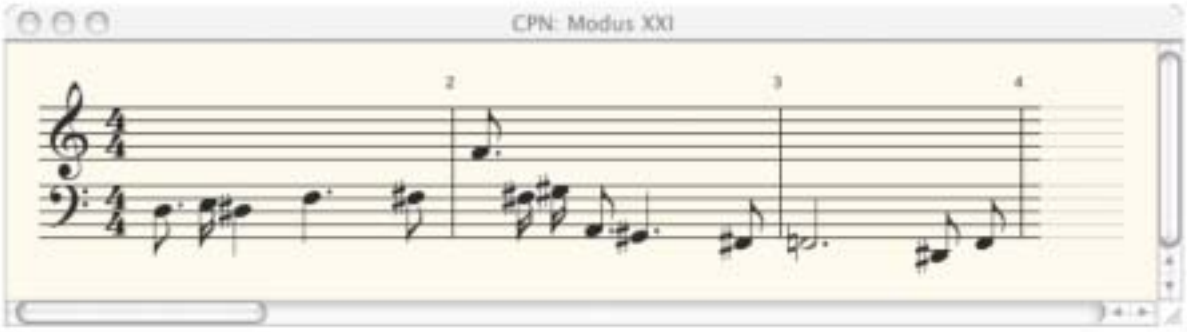
Melodías basadas en el capítulo 9 del *Modus Novus*





Melodías basadas en el capítulo 10 del *Modus Novus*





Melodías basadas en el capítulo 12 del *Modus Novus*