Capítulo II: Marco Teórico

Este trabajo de tesis abarca varios temas distintos dentro del área de la computación y también dentro del ámbito social. Para poder realizar esta tesis, se investigó a fondo cada uno de estos temas y es fin de éste capítulo explicar brevemente los fundamentos teóricos sobre los cuales ésta tesis esta basada.

2.1 MP3

El término MP3 se ha vuelto hoy en día muy común, y en especial dentro del ámbito musical y computacional. Es tan común que incluso se utiliza como una palabra, cuando en realidad son dos siglas y un número los cuales tienen un significado completamente desconocido para los millones de usuarios que utilizan el término a diario; y para todos estos usuarios su significado es tan trivial como la manera en la que el MP3 funciona. Utilizo la palabra trivial porque para la gran mayoría de los usuarios el MP3 funciona y funciona bien, sin cuestión ni dificultad.

MP3 es una manera rápida de referirse al algoritmo de compresión de audio llamado *MPEG-1 Layer III*, desarrollado primordialmente por un grupo tecnológico alemán llamado *Fraunhofer & Thompson IIS-A*, el cual comenzó a trabajar en 1987 en codificación de audio perceptivo en el marco de trabajo del proyecto EUREKA EU147, el cual trabajaba en el ámbito de Transmisión Digital de Audio comúnmente conocido en inglés como *Digital Audio Broadcasting* (DAB). Con la ayuda de la universidad de *Erlangen*, en especial del *Prof. Dieter Seitzer*, el *Fraunhofer & Thompson IIS-A* finalmente creo un algoritmo de alto nivel sumamente poderoso. Hoy en día es oficialmente estandarizado por la *International Standards Organization* comúnmente conocida como *ISO* con el nombre de *ISO-MPEG Audio Layer-III* (IS 11172-3 e IS 13818-3). El nombre MP3 se deriva directamente de la extensión que se les da a estos archivos de audio (*.mp3*).

2.1.1 Estándares MPEG

MPEG, el nombre genérico del MP3, no es solamente un estándar, sino más bien toda una familia de estándares especializados en compresión de audio y video. Las siglas MPEG son la abreviación del nombre de este grupo de estándares: *Moving Picture Experts Group*, o su traducción en español: Grupo de Expertos de Imagen en Movimiento. Este grupo fue formado en 1998 con el propósito de unificar todos los algoritmos de compresión de audio y video en uno solo; evitando así una guerra de estándares entre varias tecnologías compitiendo. Todos los estándares del grupo MPEG son utilizados para realizar compresión sobre datos audio-visuales.

La familia de estándares MPEG, está dividida en tres grandes clases: MPEG-1, MPEG-2 y MPEG-4; y estas a su vez se subdividen en ciertas clases que son llamadas *Layers* o capas; de manera que cada capa subsiguiente es más compleja que su anterior. Cada clase de MPEG y sus respectivas capas, existen para optimizar distintas tareas del mundo real; algunas de dichas tareas pueden ser la transmisión en vivo de audio o video (*broadcast*), la compresión de video y audio, o la compresión de música basada en archivos computacionales.

En este trabajo de tesis, estaremos trabajando con archivos del tipo MPEG-1 Layer III y MPEG-2 Layer III. Técnicamente, estos dos son conocidos como MP3 ya que las diferencias entre ellos son mínimas. El MPEG-1 Layer III es utilizado para frecuencias de audio de 32, 44.1 y 48 kHz. El MPEG-2 Layer III es utilizado para frecuencias de audio de 16, 22.5 y 24 kHz. A pesar de estas diferencias, cualquier programa que pueda reproducir archivos tipo MP3, podrá reproducir cualquiera de estos dos tipos de archivos y realmente no se hará diferencia ninguna.

Algo importante de mencionar, es que no debemos confundir el MPEG-1 Layer III (MP3) con MPEG-3, esa clase no existe. Al principio sí existía la clase MPEG-3 y estaba destinada a manejar video de alta calidad; sin embargo, la clase MPEG-2 demostró contar con suficiente calidad de video de tal manera que MPEG-3 fue unida con MPEG-2. Hoy en día las únicas clases existentes son: MPEG-1, MPEG-2 y MPEG-4. Para mayor información del grupo MPEG, consultar la página http://www.mpeg.org.

2.1.2 Algoritmo de Compresión MPEG-1 Layer III

Pero realmente, ¿qué son los archivos MP3? De manera muy sencilla, el MP3 es un algoritmo de alta compresión de audio el cual permite tener gran calidad de sonido disminuyendo la cantidad de información contenida en cada archivo. Existen varias técnicas de compresión de archivos las cuales prácticamente se basan en buscar redundancias en los archivos y eliminar dichas redundancias para así disminuir el tamaño del archivo. El algoritmo de compresión MP3 no se basa en encontrar redundancias dentro de los archivos de audio, sino en analizar ciertos patrones dentro de un canal de audio, los cuales son comparados con los modelos de la percepción auditiva humana. Esto es lo que se conoce como técnicas de compresión psicoacústicas. Estas técnicas, utilizadas por todo el grupo de compresión de audio MPEG, exploran las limitaciones del oído humano así como las limitantes del cerebro para procesar ciertos tipos de sonido a muy altas resoluciones. La manera en la que funcionan, es que los algoritmos de compresión de audio MPEG, guardan lo que podría ser un mapa de la capacidad auditiva humana en una tabla la cual es comparada con las entradas de canales de bits de audio deseadas. La persona que esté realizando dicho proceso es la que especifica la cantidad de bits por segundo que serán utilizados en el archivo resultante de la compresión; a esto se le conoce como el *bitrate*. Tomando en cuenta esta restricción, el algoritmo trata de remover la mayor cantidad de datos tratando de mantener la mayor calidad de audio posible. Mientras el usuario permita la mayor cantidad de bits por segundo, mayor será la calidad de audio pero a la vez mayor será el tamaño del archivo resultante de dicho proceso. Viceversa, a menor cantidad de bits por segundo, menor la calidad de audio y menor el tamaño del archivo.

Sin compresión de datos, las típicas señales de audio consisten de pedazos de 16 bits gravados en un rango mayor del doble del ancho de banda del audio actual (44.1 kHz para los discos compactos comunes). Por lo tanto al final contamos con más de 1.400 Mbits para representar solo un segundo de música estéreo con calidad de disco compacto. Utilizando el codificador de audio MPEG-1 LayerIII se pueden reducir los datos de audio originales de un disco compacto común a una relación de diez, sin perder calidad de sonido. Aproximadamente, un minuto de audio en un disco compacto equivale a 10MB de memoria, mientras que para un MP3, un minuto de audio representa aproximadamente 1MB de memoria.

2.1.3 Estructura Interna de los Archivos MP3

Internamente los archivos MP3 son algo complejos, y quizá su compleja estructura no sirva exactamente a su propósito original. Los archivos MP3 originalmente fueron diseñados y creados para poder ser transmitidos en tiempo real a través de la radio, Internet o cualquier otro dispositivo de transmisión. Evidentemente hoy en día nos podemos dar cuenta de que eso no se llevó a la realidad ya que a pesar de ser archivos de audio comprimidos, éstos siguen siendo aún mucho muy grandes para poder ser escuchados durante una transmisión en tiempo real.

Internamente los archivos MP3 están divididos en miles de pequeños fragmentos llamados *frames* o cuadros, en español. Cada cuadro contiene una fracción de segundo de información de audio. El *decoder* o decodificador, traducido al español, de MP3 se encarga de unir y reconstruir todos los cuadros para así formar un sonido final el cual puede ser una canción o un simple sonido.

Al principio de cada cuadro contamos con un *frame header* o encabezado de cuadro, en español, el cual describe la información contenida en el cuadro al que preside. Esta es una de las características únicas de este tipo de archivos, ya que normalmente todos los archivos computacionales contienen un encabezado único al principio de dicho archivo. El encabezado es utilizado para describir y proporcionar información acerca del archivo que lo contiene. Los archivos MP3 fueron diseñados de esta manera ya que permite al receptor, de una transmisión en vivo, poder entrar en cualquier momento al canal de transferencia y poder decodificar dicha transmisión desde cualquier punto, sin tener que esperar el principio de un nuevo archivo. Por esta razón los encabezados de los MP3 se encuentran al principio de cada cuadro, de esta forma, el receptor puede obtener información de cualquier cuadro antes de que éste sea recibido y así poder decodificar el archivo desde cualquier punto. Recordemos que estos archivos fueron originalmente diseñados para ser transmitidos y recibidos al aire en tiempo real.

Cada encabezado de cuadro está compuesto por 32 bits (4 bytes) de información el cual comienza con un bloque conocido como el bloque *Sync*, llamado así ya que nos permite sincronizar, en inglés *synchronize*, al reproductor con el archivo en cualquier momento de la transmisión. Este bloque consiste de once bits prendidos (doce en el caso del MPEG 2.5) y permite que los reproductores de MP3 busquen y entren en las

primeras instancias de cualquier cuadro valido, lo cual es muy útil ya que permite saltar de una sección de audio a otra rápidamente. También nos ayuda a esquivar cualquier otro tipo de información que esté contenida dentro del archivo y que no sea necesaria para la reproducción del sonido. Hay que mencionar que no es suficiente que el reproductor de MP3 encuentre un bloque *Sync* válido (once bits prendidos) y suponga que dicho bloque binario pertenece a un archivo MP3 válido, ya que teóricamente es posible encontrar en cualquier archivo binario un bloque de once bits prendidos. Por lo tanto, el reproductor debe también verificar la existencia de otros encabezados de cuadros válidos o por varios cuadros válidos en fila. El diagrama **2.1** describe gráficamente la estructura interna de un cuadro de un MP3.

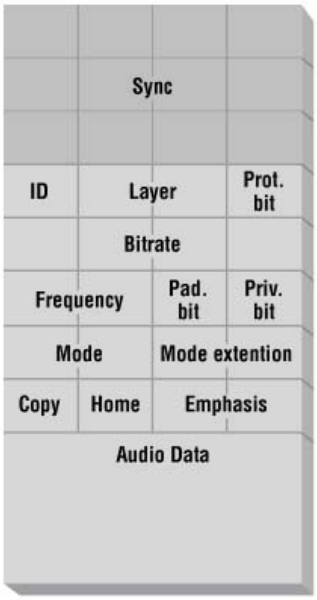

Figura 2.1
Estructura interna de un cuadro de un MP3.

Después del bloque de sincronización, continúa un bloque con longitud de dos bits, llamado el *ID Block* (bloque de identificación). Este bloque nos indica el tipo de codificación en la que el cuadro que le sigue está. El cuadro puede estar codificado en MPEG-1 o MPEG-2. Al bloque de identificación, le siguen dos bits que describen el nivel de compresión del cuadro. A este bloque se le conoce como el *Layer Block*, el cual indica el *Layer* o la capa a la que dicho cuadro está codificado. Recordemos que existen tres tipos de capas: I, II, III ó en su defecto, capa sin definición. Después del bloque de capas, sigue el *Protection Bit* (bit de protección). Si dicho bit no se encuentra

prendido, se insertará una cadena de 16 bits antes de los datos de audio contenidos en dicho cuadro.

El bloque que sigue es llamado el bloque del *Bitrate*, dicho bloque indica la cantidad de bits que en un segundo de audio o video se transmiten. Se necesitan cuatro bits ya que el rango de bits transmitidos puede ir desde 8 hasta 448 kbps; dependiendo de la versión de MPEG y de la capa a la que esté codificado dicho cuadro. Siguiente a este bloque, se encuentra el *Frequency Block* (bloque de frecuencia). Dicho bloque está formado por dos bits e indica la frecuencia de audio a la que dicho cuadro se encuentra codificado. El siguiente bloque, llamado *Padding bit*, trabaja en conjunto con los dos bloques anteriores. Este bloque formado, por un solo bit, se encarga de que cada cuadro satisfaga exactamente los requerimientos del *bitrate*. Por ejemplo, un 128 kbps Layer III canal de bits a 44.1 kHz puede acabar teniendo ciertos cuadros de 418 bytes y otros de 417 bytes. En los cuadros que se encuentren 417 bytes, el *padding bit* (bit de acolchonamiento) estará prendido para compensar dicha discrepancia.

El siguiente bloque, llamado *Private bit*, descrito por solo un bit es solamente utilizado por ciertas aplicaciones a manera de detonador. Posterior a dicho bloque, se encuentra un bloque llamado *Channel Mode Block*. Este bloque se refiere al estatus (estéreo/mono) del cuadro que preside, nos indica el tipo de canal de audio que dicho cuadro utiliza. El canal de audio puede ser: *Stereo* (Estéreo), *Joint Stereo* (Estéreo Conjunto), *Dual Channel* (Canal Dual) o Mono (un solo canal). Esta información es utilizada por el siguiente bloque llamado *Mode Extension Block*, compuesto por dos bits. En caso de que el cuadro utilice *joint stereo* como canal de audio, este bloque le indica al decodificador exactamente como manejarlo. El rango completo de la frecuencia de un archivo MPEG-1 es dividido en bandas; existen 32 bandas. Para las capas I y II estos dos bits determinan el rango de frecuencia (bandas) donde se aplica estéreo con intensidad. Para la capa III estos dos bits determinan el tipo de *joint stereo* que será utilizado. Existen dos tipos de *joint stereo*: *Intensity Stereo* (Estéreo con Intensidad) y *M/S Stereo* (Estéreo M/S). El rango de frecuencia se determina con un algoritmo de descompresión.

El siguiente bloque corresponde al *Copyright*, el cual se refiere a los derechos de autor de dicho archivo. El bloque no contiene información acerca de los derechos del autor de dicho archivo, sino más bien funciona como una bandera, similar a la utilizada en los discos compactos, la cual solamente indica si el archivo esta protegido por ciertos derechos de autor. Dicho bloque esta compuesto por solamente un bit al igual que el

bloque que le sigue. Este bloque llamado *Original Block*, (bloque de originalidad) indica si el archivo es original o si es una copia del original.
Finalmente contamos con un bloque de dos bits llamado *Emphasis Block*. Este bloque es obsoleto y raramente utilizado. Le dice al decodificador que el archivo debe de ser desenfatizado, o sea, el decodificador debe de volver a ecualizar el sonido después de una supresión de ruido tipo Dolby.
El decodificador, al terminar de leer todo el encabezado, avanza finalmente a la información de audio contenida en dicho cuadro. Este proceso se repite miles de veces, una para cada cuadro existente en dicho archivo MP3.

La tabla **2.2** representa el contenido del encabezado de un cuadro de un archivo MP3. La tabla presenta los cuatro bytes que forman el encabezado. Cada letra representa uno de los trece diferentes campos, descritos anteriormente, contenidos dentro del encabezado.

Tabla **2.2**. Los cuatro bytes del encabezado de un cuadro de un archivo MP3.

Byte 1	Byte 2	Byte 3	Byte 4
AAAAAAAA	**AAABBCCD**	**EEEEFFGH**	**IIJJKLMM**

La tabla **2.3**, contenida en el **Apéndice B**, describe a detalle cada una de las características y propiedades de cada uno de los trece campos contenidos en el encabezado de cada cuadro de un archivo MP3.

2.1.4 Etiquetas ID3

Una de las características más fascinantes de los archivos MP3, es la capacidad de poder almacenar gran cantidad de datos relacionados con la información de la pista contenida en dicho archivo. Tal información puede constar del título de la pista, el autor, el género, el álbum al que pertenece, etc. Toda esta información es almacenada

dentro del archivo utilizando metadatos en la forma que hoy en día conocemos como *ID3 Tags*, o su traducción al español, etiquetas ID3. La gran mayoría de los reproductores de MP3 tienen la capacidad de leer tales etiquetas y desplegar su información en la interfase gráfica del reproductor, facilitando así al usuario el acceso a la información relacionada con pista que está escuchando en ese momento. Esta información se puede agregar o modificar al momento de la codificación de la pista, o incluso después de su codificación; en cualquier momento deseado.
En realidad, las etiquetas ID3 son parte de una especificación de etiquetado descrita por la misma organización que las inventó; dicho grupo es conocido también como ID3 (http://www.id3.org).

Existen dos variantes de la especificación ID3: ID3v1 e ID3v2, y aunque las diferencias entre ambos son grandes, casi todos los reproductores de MP3 modernos son capaces de manejar archivos con etiquetas en cualquiera de los dos formatos. De los dos tipos de etiquetas, las ID3v2 son superiores por tres razones principales:

- Las etiquetas ID3v2, a diferencia de las ID3v1, pueden contener hasta 256 MB de información en ellas. Lo que significa que hay suficiente espacio para poder almacenar imágenes, la letra completa de la pista, preferencias de ecualización, etc.

- La información contenida en las etiquetas ID3v2, se encuentra al principio del archivo; por lo contrario, la información de las etiquetas ID3v1 se encuentran al final del archivo. Es muy útil que la información esté contenida al principio del archivo (recordemos que los archivos MP3 fueron diseñados para ser transmitidos al aire en tiempo real) de manera que los reproductores pueden desplegar la información de la pista que se está reproduciendo desde el principio de la transmisión, y no tener que esperar al final de la transmisión para poder desplegar su información.

- La especificación ID3v2 es abierta y flexible de manera de que dicha especificación puede ser extendida para satisfacer necesidades aún no vistas.

Son muchas las posibles aplicaciones que se le pueden dar a la especificación ID3v2. Una de las más importantes que se está comenzando a explorar, es la capacidad que ésta especificación le da a los artistas o disqueras de almacenar dentro del archivo información relevante a los derechos de autor, términos de utilización, certificado de autenticidad, etc. Incluso se están empezando a explorar la posibilidad de incluir seguridad criptográfica dentro de los archivos para que éstos solo puedan ser reproducidos utilizando ciertos certificados de utilización.

A pesar de que la utilización de las etiquetas ID3v2 parece ser solamente benéfica, existe una desventaja muy importante por considerar. Esta desventaja se presenta principalmente en los reproductores antiguos de MP3, y los reproductores que no fueron construidos dentro de la especificación ISO de MP3. Recordemos que las etiquetas ID3v2 se encuentran al principio del archivo, y que tales etiquetas pueden contener hasta 256 MB de información. Entonces, un reproductor de MP3 debería solamente evitar toda la información contenida en la especificación ID3v2, y que por supuesto no entiende, y saltar directamente a los datos de audio para comenzar a decodificar la información. Pero entonces la pregunta que surge es: ¿qué tan lejos debe el reproductor buscar el encabezado del primer cuadro válido del MP3 antes de decidir que el archivo que está recorriendo en realidad no es un archivo MP3? Desafortunadamente la especificación ISO en el tema es realmente vaga. El problema entonces es que cada desarrollador genera su propio criterio de búsqueda del primer cuadro válido dentro del MP3. Algunos piensan que el reproductor debería recorrer todo el archivo hasta encontrar el primer cuadro válido. Otros consideran que después de un cierto número de bytes, el reproductor se debe dar por vencido y dejar de buscar, concluyendo así, que el archivo no es un MP3 válido para reproducir.
Para este trabajo de tesis, estaremos trabajando con ambos tipos de etiquetas; ID3v1 e ID3v2.

2.1.5 El Impacto de los Archivos MP3 en la Industria Musical

Los archivos MP3 han tenido un impacto en nuestra sociedad más allá de lo previsto hace algunos años. Quizá en 1996, el termino MP3 era para muchos de nosotros algo completamente desconocido. Incluso a principios de 1997, para muchas personas, los archivos MP3 no tenían mucho futuro ya que eran descritos como ilegales y difíciles de manejar; algo que solo la gente experta en computación podía utilizar. Nunca se pensó que estos archivos llegaran a superar a los discos compactos como el formato número uno de música para la gran cantidad de usuarios de la industria. Hoy en día los MP3 son algo sumamente común y apenas comenzamos a explotar el gran potencial que la música digital tiene.

En Julio de 1997, la Federación Internacional de Industrias Fonográficas (IFPI) estimó que alrededor de tres millones de pistas de audio eran intercambiadas por usuarios a través de Internet cada día, las cuales sin permiso de utilización ni de copia de los dueños de los derechos de autor. La RIAA (*Recording Industry Association of America*) expresó haber perdido aproximadamente la cantidad de $10 billones USD a través de la música pirata en Internet en 1998. Pero realmente no son solo las compañías disqueras las cuales pierden; sino también los artistas, dueños de tiendas de discos, las fábricas de impresión de discos compactos, y la lista sigue. Muchas disqueras y artistas piden a diario a la RIAA, que terminen con los sitios de Internet piratas los cuales distribuyen su música sin su autorización.

A pesar de esto, tenemos el otro lado de la moneda. Existen muchos artistas los cuales publican su música o parte de ella en sus propias páginas de Internet. Incluso muchos de ellos apoyan el intercambio de archivos ya que se ha llegado a demostrar que dicha distribución de material discográfico ayuda a que los artistas se den a conocer en lugares en los que sus discos no se venden. La razón principal de esto, es que muchos de los artistas que hacen música no tienen un contrato con alguna compañía disquera y evidentemente tampoco cuentan con el capital para poder distribuir su trabajo por todo el mundo, como lo hacen las grandes disqueras. Dichas disqueras no apoyan a gran cantidad de artistas afirmando que su trabajo no alcanzaría las ventas necesarias. A raíz de esto, los artistas hacen lo que pueden para darse a conocer; esto incluye publicar su trabajo en Internet. Entonces nos encontramos con la cuestión de que hasta cierto punto,

las disqueras son las que escogen la música que se escucha y la música que no se escucha; y a esto le podemos sumar que con los precios tan altos a los que la música legal se vende, de cierta forma las disqueras también deciden quien escucha música y quien no.
No es fin de esta tesis generar polémica ni investigar el tema a fondo, simplemente se describen los hechos como son de manera neutral. De la misma manera el contexto social de esta tesis describe una posible solución a algunos de los conflictos generados por la distribución de archivos digitales de audio a través de Internet.

Las características descritas previamente de los archivos MP3, así como la popularidad, su potencial, la facilidad para trabajar con ellos, y el impacto que estos archivos tienen sobre las disqueras y sobre nuestra forma de ver y escuchar música, fueron algunas de las razones por las cuales se escogieron los archivos MP3 para ésta tesis. Existen muchos otros archivos de audio digitales los cuales no describiremos detalladamente, pero si es de interés mencionarlos y compararlos con los archivos utilizados en esta tesis. La tabla **2.4**, contenida en el **Apéndice B**, es una tabla comparativa entre los archivos MP3 y algunos de los archivos de audio más populares hoy en día. La tabla fue obtenida del libro MP3: The Definitive Guide, escrito por el autor Scott Hacker; la bibliografía completa del libro se encuentra descrita en la **Bibliografía** de éste documento de tesis.

2.2 Patrón de Diseño MVC

A través del tiempo, ha ido creciendo la industria productora de software. Pero no solo eso, sino que cada vez son más las personas que están aprendiendo diferentes lenguajes de programación y por lo tanto, también se encuentran produciendo software. Aunque la producción de software desarrollada por personas físicas o pequeñas empresas es realmente distinta a la producción a gran escala de los grandes emporios desarrolladores de software, todos ellos tienen algo en común: todos ellos presentan ciertos patrones de diseño de software. Uno de esos patrones de diseño es el *Model View Controller* (MVC), o en español, Modelo Vista Controlador. Existen muchos patrones de diseño, pero para esta tesis se decidió trabajar con el patrón de diseño MVC. Para ésta parte, se utilizó el libro Head First Design Patterns escrito por los autores *Eric Freeman, Elisabeth Freeman, Kathy Sierra* y *Bert Bates* como fuente bibliográfica primaria. La bibliografía completa de dicho libro se encuentra descrita en la **Bibliografía** de éste documento de tesis.

2.2.1 Patrones de Diseño

Los patrones de diseño son una de las herramientas más poderosas que hoy en día tenemos a nuestra disposición para ayudarnos a desarrollar software de mayor calidad. Contrario a lo que se cree, los patrones de diseño no son nada nuevo. Podemos encontrar patrones de diseño no solamente en la industria del software, sino en cualquier industria en la cual la tarea de diseñar esté involucrada. Podríamos decir entonces, que podemos encontrar patrones de diseño casi en cualquier parte. Todo lo que el hombre produce o desarrolla, necesita un cierto nivel de diseño; y son esos procesos de diseño los que a través del tiempo se manifiestan en patrones. Patrones los cuales son estudiados y perfeccionados para que se puedan convertir en estándares, lo cual nos ayuda a asegurar la calidad del producto o servicio que se este desarrollando.

Pero entonces ¿qué es un patrón de diseño formalmente? Formalmente podríamos decir que un patrón de diseño es una solución de alta calidad para un problema recurrente de diseño, es decir, es una solución de gran calidad para problemas de diseño que se repiten constantemente. En contexto computacional, podríamos decir que un patrón de

diseño nos provee de un esquema para refinar los subsistemas o los componentes de un software y la relación que existe entre ellos. Nos describe una estructura recurrente de componentes que se comunican, la cual resuelve un problema de diseño general con un contexto particular. Todos los patrones de diseño proveen alguna manera de que aquellas partes del sistema que cambian, lo hagan totalmente independiente sin afectar el resto de la aplicación.

No importa que tan grande o pequeño sea el software por desarrollar, los patrones de diseño son altamente utilizables a cualquier nivel. Inclusive, un patrón de diseño bien delimitado, puede llegar a convertirse en un modelo de diseño particular, o incluso en un mecanismo de diseño muy específico y funcional. Todos los patrones de diseño son completamente independientes del lenguaje de programación que se desee utilizar, pero si están estrechamente relacionados con los principios de diseño orientado a objetos. Esto debido a que los patrones de diseño parten de la premisa de separar los diferentes elementos u objetos de cualquier sistema, de manera de que cada uno de ellos trabaje como una entidad aislada la cual se comunica con otras entidades a través de ciertas interfases. Dichas interfases se convierten en componentes de abstracción los cuales logran la interacción del resto de los objetos; aún así, incluso los componentes de abstracción actúan como una entidad u objeto independiente. Para un mejor entendimiento de la estructura y funcionamiento de los patrones de diseño, es necesario conocer las bases del diseño orientado a objetos.

2.2.2 Principios de los Patrones de Diseño

Los patrones de diseño surgieron a partir de la observación. Dichas observaciones llevaron a la creación de algunas reglas o principios sobre los cuales los patrones de diseño se basan. Hoy en día, dichos principios pueden ser considerados como las leyes que rigen a los patrones de diseño. Algunos de los principios más importantes son:

- **Identifica los aspectos de tu aplicación que varían, y sepáralos de los que aspectos que se mantienen iguales.**

  Considerado como el principio más importante, nos indica que debemos encapsular todas las partes de nuestra aplicación que sí cambian y separarlas de

las que no cambian. De esta manera, podemos más tarde alterar o extender las partes de la aplicación que cambian sin afectar al resto de la aplicación.

- **Programa para una interfase, no para una implementación.**
  Entiéndase por interfase, como la descripción del comportamiento de algo. Este principio nos indica que es más importante describir el comportamiento, que en sí, describir la implementación de cambios en dicho comportamiento.

- **Favorece la composición sobre la herencia.**
  Al describir cierto objeto, es mucho mejor si lo relacionamos con sus comportamientos a través de la composición (tiene un) en vez de relacionarlo a través de la herencia (es un). Esto nos dará mucha más flexibilidad ya que nos permite encapsular toda familia de algoritmos que describan dichos comportamientos, y también nos da la posibilidad de cambiar el estado de cualquier comportamiento del objeto cuando el sistema esté corriendo.

- **Las clases deben de estar abiertas para ser extendidas, pero cerradas para ser modificadas.**
  Nos interesa que nuestras clases sean fáciles de extender para poder agregar nuevos comportamientos. No deseamos modificar los comportamientos ya existentes.

- **Depende de abstracciones, no de clases concretas.**
  Los componentes de alto nivel no deben de depender directamente de los componentes de bajo nivel. Al igual que los componentes de bajo nivel no deben de depender directamente de los componentes de alto nivel. Debe de existir un componente de abstracción el cual comunique a ambos niveles, de manera que ninguno de ellos dependerá directamente del otro.

- **Comunícate solo con los componentes inmediatos.**
  Al diseñar un sistema, debemos de cuidar que el número de clases que interactúan con un cierto objeto sea el mínimo; incluso debemos de cuidar la manera en la que se comunicarán. Esto nos da la facilidad de que si realizamos

cambios en una parte del sistema, estos no se reflejen en todas las demás partes del sistema, de manera que el grado de dependencias entre las clases es muy pequeño.

- **No me llames, yo te llamo.**
  Este principio es muy parecido al principio referente a la dependencia sobre abstracciones. Permite que los componentes de bajo nivel se integren al sistema en cualquier momento, pero solamente los componentes de alto nivel determinan cuando los mandan llamar y de que manera. Los componentes de bajo nivel nunca deberían de llamar a los componentes de alto nivel.

- **Una clase solamente debe de tener una responsabilidad.**
  Cada clase u objeto tiene una razón de ser. Debemos de mantener dicha razón, de manera que cada clase se dedica a una cierta tarea específica, pueden ser varias tareas pero dentro del mismo tema o contexto. Si una clase se sale de contexto y por lo tanto tiene más de una responsabilidad, entonces deberíamos desarrollar una nueva clase que se encargue de la nueva responsabilidad. Cada quien a lo suyo.

2.2.3 Tipos de Patrones de Diseño

Existen muchos tipos de patrones de diseño. Cada uno de ellos enfocado a realizar un diseño en específico. Se presenta a continuación una lista de los patrones más utilizados y una breve descripción de su función:

- **Patrón Decorador**
  Envuelve un objeto para proveer un nuevo comportamiento.

- **Patrón de Estado**
  Encapsula comportamientos basados en cambios de estado y utiliza técnicas de delegación para cambiar entre comportamientos.

- **Patrón de Iteración**

  Provee métodos para recorrer y explorar alguna colección de objetos sin exponer su implementación.

- **Patrón de Fachada**

  Simplifica la interfase de una serie de objetos.

- **Patrón de Estrategia**

  Encapsula comportamientos intercambiables y utiliza técnicas de delegación para decidir cual de ellos utilizar.

- **Patrón Proxy**

  Envuelve un objeto para controlar su acceso.

- **Patrón del Método de Construcción**

  Ayuda a los objetos a determinar, de manera concreta, qué otros nuevos objetos deben crear; en realidad instancias de esos objetos.

- **Patrón Adaptador**

  Envuelve un cierto objeto y provee una nueva y diferente interfase de acceso a él.

- **Patrón de Observación**

  Le da capacidad a los objetos de ser notificados en caso de que el estado del sistema o cualquier otro objeto cambien.

- **Patrón del Método de Plantilla**

  Ayuda a los objetos a decidir cómo implementar ciertos pasos en un algoritmo.

- **Patrón de Composición**

  Los clientes tratan a colecciones de objetos y a objetos individuales de la misma manera.

- **Patrón de Unicidad**

  Asegura que solamente se creará una sola instancia de cierto objeto.

- **Patrón de Creación Abstracta**

  Le permite a un cliente crear familias de objetos sin tener que especificar sus clases concretas.

- **Patrón de Comandos**

  Encapsula cualquier petición como un objeto.

2.2.4 MVC: Patrón de Diseño Compuesto

Hasta ahora, solamente hemos descrito a varios patrones de diseño trabajando completamente independientes uno del otro. Pero una de las herramientas más poderosas cuando utilizamos el diseño orientado a objetos es cuando varios patrones de diseño trabajan en conjunto. A pesar de que podemos ver y definir cada uno de los patrones individualmente, son mucho más poderosos cuando los combinamos para desarrollar un diseño orientado a objetos. A esto se le llama *Compound Pattern*, o en español, patrón compuesto. Un patrón compuesto es aquel, que como dice su nombre, se encuentra integrado por dos o más patrones los cuales se combinan para llegar a una solución que resuelve un problema recurrente o general. Uno de los patrones compuestos más famosos y utilizados hoy en día es el famoso *Model View Controller* (MVC) del que hablábamos anteriormente. Por esta razón es importante conocer y entender ciertos patrones de diseño antes de involucrarnos con MVC. Este patrón compuesto utiliza primordialmente tres patrones de diseño descritos anteriormente: el Patrón de Observación, el Patrón de Estrategia y el Patrón de Composición. Más adelante explicaremos a detalle como es que cada uno de estos tres patrones encaja dentro del MVC.

El MVC es un patrón compuesto el cual nos ayuda a diseñar un sistema dividiéndolo en tres partes:

- **Modelo**

  El Modelo es la parte que se encarga del almacenamiento de datos, cambios de estado, lógica aplicativa, etc. Es la parte del sistema en donde se llevan a cabo los cambios de estado del mismo. El Modelo es la única parte del sistema que debe de comunicarse con la base de datos, algún índice, el disco duro, etc. El Modelo es el que conoce todas las reglas de comportamiento del sistema; la parte que realiza los procesos que provocan un cambio de estado en dicho sistema. El Modelo es totalmente independiente del Controlador y de la Vista, y está completamente desinteresado en la existencia de ambos.

- **Vista**

  Esta parte es la encargada de todo lo referente con la interfase que interactúa directo con el usuario, ya sea de manera gráfica, electrónica, mecánica o de cualquier otro tipo. Es el encargado de presentar al usuario los cambios de estado que se registren en el sistema. Prácticamente, la Vista nos da una presentación del estado en el que el Modelo se encuentre. Esta parte recibe el estado actual del Modelo a través del Controlador, indirectamente. El Controlador pone los datos del Modelo en algún lugar en el que la Vista los pueda encontrar y desplegar. Al igual que el Modelo, la Vista es completamente independiente de los otros dos elementos.

- **Controlador**

  Como su nombre lo dice, es el encargado de controlar las transacciones entre la Vista y el Modelo y viceversa. Podríamos decir que el Controlador es la capa de en medio. Esta parte se encarga de recibir la entrada del usuario y traducirla de manera que el Modelo la pueda entender. El Controlador le dice al Modelo que se debe de actualizar y hace que el nuevo estado del Modelo esté accesible para que la Vista lo presente; y al igual que las dos partes anteriores, el Controlador es completamente independiente.

A pesar de que cada una de las tres partes de este patrón de diseño son altamente independientes, todas ellas interactúan de una manera muy fluida. Esto debido a que cada parte se encuentra perfectamente bien delimitada y cada una realiza una tarea muy específica. Para entender mejor su funcionamiento y la manera en la que las tres partes del MVC interactúan, supongamos el siguiente escenario: tenemos a un cierto usuario el cual esta utilizando cierto software. El usuario interactúa directamente con la Vista. La Vista es la ventana del usuario hacia el Modelo. Cuando el usuario realiza alguna acción, supongamos que presiona un botón, la Vista le dice al Controlador que el usuario presionó cierto botón. Es responsabilidad del Controlador interpretar y manejar dicha acción. La Vista solamente le avisó al Controlador que el usuario presionó un botón y le informó cuál botón fue presionado. El Controlador recibe dicha acción y la interpreta. Es responsabilidad del Controlador averiguar que significa que el usuario haya presionado cierto botón, y cómo debe el Modelo ser manipulado basado en dicha acción. Parte del trabajo del Controlador, es pedirle a la Vista que se actualice. Una vez que el Controlador recibe la acción de la Vista, es posible que el Controlador le tenga que decir a la Vista que necesita cambiar en ciertos aspectos. Por ejemplo, si se presiona cierto botón, quizá el Controlador le tiene que avisar a la Vista que necesita desactivar otros botones o menús, incluso quizá desplegar un mensaje de que la petición del usuario está siendo procesada.

Es tarea del Modelo notificar a la Vista, a través del Controlador, que el estado del sistema ha cambiado y que la Vista necesita actualizarse. Dicho cambio puede darse por alguna acción del usuario o por algún proceso interno del sistema. Una vez que la Vista es notificada de esto, le pide al Modelo la información del cambio de estado del sistema que necesita desplegar para el usuario. Este ciclo se repite cada vez que algún evento cambia el estado del sistema.

Anteriormente, mencionamos que el MVC utiliza tres patrones de diseño: el Patrón de Observación, el Patrón de Estrategia y el Patrón de Composición. Cada uno de estos patrones está diseñado para realizar una tarea en específico, pero si los combinamos de manera correcta, obtenemos una herramienta de diseño orientada a objetos muy poderosa. Pero, ¿cómo encajan estos tres patrones dentro del MVC? Comencemos con el Modelo. El Modelo implementa el Patrón de Observación para mantener a los objetos actualizados cuando cualquier cambio en el sistema ocurra. Utilizando el Patrón de Observación, permite mantener al Modelo completamente

independiente de la Vista y del Controlador. Inclusive nos permite utilizar varias Vistas para el mismo Modelo, o Vistas múltiples al mismo tiempo.
La Vista y el Controlador implementan el Patrón de Estrategia. La Vista es un objeto el cual es configurado con una cierta estrategia. El Controlador provee dicha estrategia. La Vista solamente se preocupa por los aspectos visuales de la aplicación y le delega al Controlador cualquier decisión acerca del comportamiento de la interfase gráfica. Utilizando el Patrón de Estrategia ayuda a mantener a la Vista independiente del Modelo ya que es el Controlador quién interactúa con el Modelo, y la Vista no sabe como se realiza dicha tarea.
La Vista, a su vez, implementa el Patrón de Composición para poder administrar todas las ventanas, botones, menús, y cualquier otro componente de la interfase gráfica. Toda interfase gráfica, está compuesta por varios elementos distintos: botones, campos de texto, menús, etc. Cada uno de esos componentes es un objeto, y todos juntos forman una composición de componentes o una colección de objetos. Cuando el Controlador le dice a la Vista que necesita actualizarse, el Controlador solamente se debe comunicar con un solo componente, y ese componente debe de encargarse de comunicárselo al resto de los componentes individualmente.

2.2.5 MVC en Aplicaciones Web

El patrón de diseño MVC existía mucho antes de que la *World Wide Web* llegara. Inicialmente, MVC fue diseñado para simplificar sistemas que contaran con una interfase gráfica muy compleja. Pero no tardó mucho antes de que el MVC comenzara a ser utilizado en aplicaciones web. Los desarrolladores de dichas aplicaciones adaptaron el modelo MVC para que encajara dentro del modelo navegador/servidor que se utiliza en la mayoría de aplicaciones web. Esta adaptación es conocida simplemente como *Model 2*, o en español Modelo 2. Específicamente refiriéndonos al lenguaje de programación Java, el *Model 2* utiliza las tecnologías de *Servlets* y *JSP* para alcanzar el mismo nivel de separación entre el Modelo, la Vista y el Controlador.
Uno de los atributos más característicos del MVC es que la Vista sería notificada automáticamente de los cambios que se lleven a cabo en el Modelo. Esto presenta una restricción en contexto de web dinámico, ya que la parte de la Vista de la aplicación se

encuentra en el navegador de Internet y éste no puede ser actualizado automáticamente cuando el Modelo del sistema cambie de estado. La actualización de la vista solamente se puede realizar a través de una petición enviada por la Vista a través del Controlador hacia el Modelo y de regreso. Pero si la Vista no hace dicha petición, el Modelo no le puede avisar al Controlador del cambio de estado en el sistema y el Controlador a su vez no puede enviarle a la Vista la información que debe desplegar. La razón de esto, es que las aplicaciones web que utilizan el modelo navegador/servidor funcionan de manera asíncrona, es decir, cada vez que se realiza una petición al servidor a través del navegador, dicho navegador se conecta con el servidor, realizar la petición deseada, recibe una respuesta y se desconecta. Una vez desconectado el servidor no le puede pedir al navegador que se vuelva a conectar. Cada petición del navegador al servidor es una conexión distinta.

Ahora, ¿cómo exactamente actúa el MVC en un contexto de web dinámico? Para explicar esto nos ayudaremos gráficamente del diagrama **2.5** el cual describe dos cosas: el comportamiento de cualquier petición hecha por el navegador hacia el servidor web y su respuesta, y describe qué parte del sistema corresponde a cada una de las tres partes del MVC. Para dicha explicación, utilizaremos las dos poderosas herramientas de web dinámico del lenguaje Java: *Servlets y JSPs*.

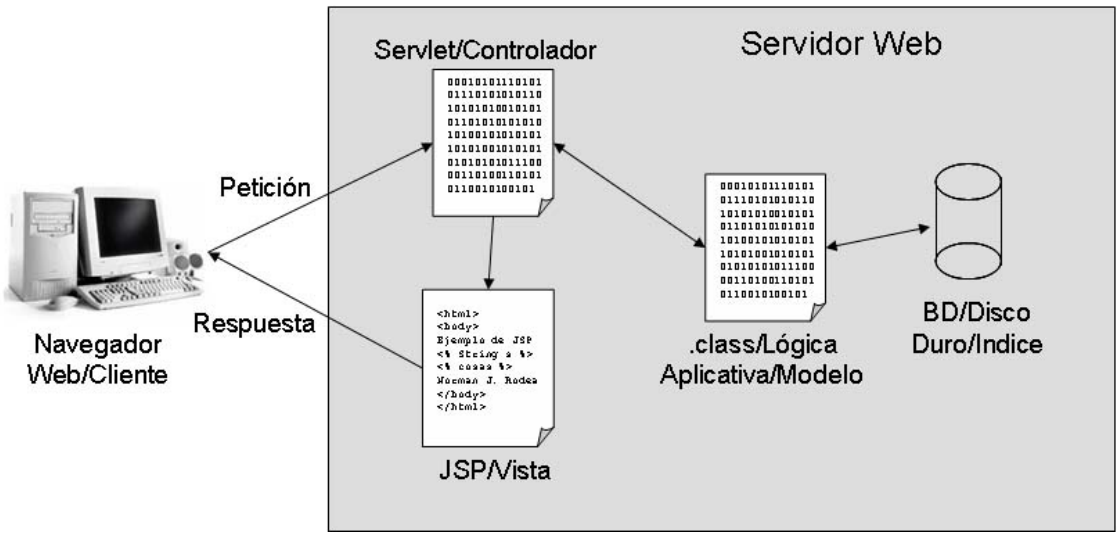

Figura 2.5

MVC aplicado a una aplicación web.

¿Que es lo que está ocurriendo en el sistema descrito por el diagrama **2.5**? Todo comienza cuando el usuario realiza una petición a través del navegador de Internet. Esto típicamente involucra el envío de algunos datos introducidos por el usuario. Un *servlet* recibe dicha petición con los datos enviados, analiza la petición y los datos para decidir lo que se tiene que hacer. El *servlet* actúa como el Controlador, procesa la petición y los datos enviados, los interpreta y se los pasa al Modelo para que este realice las operaciones necesarias y cambie el estado del sistema. El Modelo es representado por las clases típicas de Java. Todos los archivos *.class* (una clase típica de Java), representan la parte del Modelo en el MVC. Una base de datos, índices, disco duro, etc. también son parte del Modelo.

La Vista es representada por los *JSPs*. La única responsabilidad de un *JSP* es generar la página web que presenta la vista del Modelo. Esta página es la que se le devuelve al navegador como resultado de su petición. El navegador se desconecta y esto no vuelve a ocurrir hasta que el navegador vuelva a realizar una petición al servidor web.

Establecido lo anterior, podemos concluir entonces que para cada petición que el navegador pueda realizar, es decir, para cada página que tenga la aplicación web, existirá un *servlet* (Controlador), un .class (Modelo) y un *JSP* (Vista) correspondientes.

Como nos pudimos dar cuenta, las dos grandes ventajas de utilizar el patrón compuesto de diseño MVC son: alto grado en reutilización de código y alto grado de independencia en cada módulo de nuestro sistema. Estas dos son las ventajas formales e inmediatas, pero en realidad las ventajas de utilizar este patrón de diseño son infinitas.

2.3 Motores de Búsqueda

Es increíble cómo en menos de una década los motores de búsquedas han cambiado completamente la manera en la que nosotros buscamos y obtenemos la información que deseamos. Hasta hace poco, para encontrar cualquier tipo de información, necesitábamos buscar en los libros o enciclopedias que tuviéramos en casa; y si la información obtenida no era suficiente, teníamos que ir a una biblioteca y buscar cualquier tipo de publicación referente a lo que buscábamos. Eso en realidad tomaba mucho tiempo, tiempo valioso que hoy en día podemos aprovechar para otras cosas. Actualmente, la búsqueda de información se ha reducido a unos simples *clicks*, introducir nuestra búsqueda y listo. Podemos obtener en cuestión de segundos la información que necesitamos. Esto es posible gracias a los grandes avances tecnológicos, pero principalmente, se debe a los grandes avances de investigación en un área particular de las ciencias de la computación: "Almacenamiento y Recuperación de Información". Cualquier motor de búsqueda tiene sus raíces en ésta disciplina.

En realidad ésta disciplina es relativamente nueva. En el mes de septiembre de 1966, la revista norteamericana *Scientific American* publicó un artículo llamado Information Storage and Retrieval, escrito por Ben-Ami Lipetz. Este artículo es considerado como la primera publicación oficial del tema, y fue justo ahí cuando comenzó la gran historia de la recuperación de información que inclusive hoy en día se sigue escribiendo.

En éste artículo, el autor nos describe cómo las más avanzadas tecnologías de la época, solamente podían manejar tareas o búsquedas sencillas y rutinarias. El autor concluye, que no veremos avances significativos en dicha tecnología hasta que tengamos un entendimiento mucho más profundo del cerebro humano y de la manera en que dicho órgano almacena, procesa y recupera la información. Es obvio que no hemos podido llegar a tal extremo; y quizá las predicciones de Lipetz no sean tan atinadas o más bien algo extremistas, ya que es bastante claro para nosotros, los que vivimos a principios del siglo XXI, que sí hemos tenido avances significativos en el almacenamiento y recuperación de la información. Quizá no hemos alcanzado el grado de avance tecnológico deseado en dicho teme, pero creo que cada vez nos acercamos más. Esto, a pesar de que nuestro entendimiento del funcionamiento del cerebro humano sigue siendo muy pobre.

Evidentemente mientras más avanza la tecnología, son mayores las distintas formas con las que podemos representar la información. Al principio de la era computacional solamente teníamos unos y ceros. Más adelante fuimos capaces de utilizar palabras y textos completos. Hoy en día, podemos representar información en un sin fin de maneras distintas: audio, video, texto, imagen, diferentes estructuras de datos, etc. Gracias a esto, el almacenamiento y la recuperación de información, se han vuelto quizá algo más extensos y distintos de cuando Lipetz publicó su artículo. Esto es de hecho bastante común. Inclusive Karl Popper, un filósofo científico del siglo XX, nos decía que cuando formulamos teorías para resolver algún problema, en realidad no estamos resolviendo el problema, si no más bien lo estamos dividiendo en subproblemas más pequeños los cuales nos llevarán a la formulación de otras teorías. El tema de la recuperación de la información quizá esté pasando por esta etapa, ya que como nos podemos dar cuenta, los motores de búsquedas están dejando de ser genéricos y se están especializando en un tipo de dato o de búsqueda. Tomemos el famoso buscador *Google*. Al principio, *Google* solamente realizaba búsquedas sobre páginas web y todo el contenido dentro de ellas. Hoy en día, *Google*, cuenta con buscadores sumamente específicos y por su puesto su buscador principal el cual es genérico. *Google* cuenta con un buscador de páginas web, un buscador de imágenes, un buscador para comprar cualquier artículo por la red, un buscador de grupos de discusión, un buscador de publicaciones académicas, un buscador de noticias, entre otros. De esta manera dicha empresa se está encaminando de lo general a lo específico generando mejores resultados en sus búsquedas.

Es a partir de ésta idea que éste trabajo de tesis toma forma. Se decidió enfocar el motor de búsqueda de esta tesis hacia un tipo de dato en específico: archivos de audio, y en particular, archivos de audio MP3. Fue uno de nuestros fines desarrollar un buscador de archivos de audio MP3 eficaz, eficiente y específico. El cual se especializara en solamente un tipo de dato y de esta manera lograr una recuperación de información sumamente especializada.

2.3.1 Datos vs. Información

Primero que nada, necesitamos definir claramente la diferencia en entre datos e información. Los datos son la representación lógica de la realidad: números, letras, sonidos, imágenes, etc. La información está compuesta por datos a los cuales se les ha asignado un significado. El análisis de dichos datos es lo que produce información. En la vida real, lo verdaderamente útil es la información y no los datos en sí.
Es importante distinguir dicha diferencia, ya que antes de poder realizar cualquier búsqueda de cualquier tipo de información, dicha información debe de estar representada computacionalmente en algún tipo de dato. Esto nos servirá ya que si conocemos el tipo de dato, su estructura y su clasificación, será mucho más fácil realizar búsquedas y calificar los resultados.

2.3.2 Datos Estructurados, No Estructurados y Semiestructurados

Cualquier tipo de dato se puede clasificar dentro de los datos estructurados, dentro de los no estructurados, o dentro de los semiestructurados. Cada una de las clasificaciones mencionadas anteriormente, cuenta con ciertas características únicas.

- **Datos Estructurados**
  Son aquellos datos los cuales se encuentran fuertemente definidos, y cuentan con una estructura conocida y rígida. Por lo general estos datos son asociados con una base de datos.

- **Datos No Estructurados**
  Son aquellos datos que no poseen definiciones de tipos. Son datos los cuales no están organizados de acuerdo a algún patrón. Ejemplos de este tipo de datos son hojas de texto sin ninguna estructura, imágenes, audio, etc.

- **Datos Semiestructurados**

  Cualquier tipo de dato que se encuentre entre los estructurados y los no estructurados puede se considerado como semiestructurado. Son datos los cuales se encuentran pobremente especificados. El ejemplo más claro de este tipo de datos son los documentos *XML*.

Es importante mencionar y definir claramente estos tres tipos de datos ya que en ésta tesis se trabajó con los tres. Los archivos de audio MP3 son el ejemplo de los datos no estructurados. Si tomamos dichos archivos tal y como están, sin etiquetas ID3 y sin ningún tipo de identificador, las cadenas binarias internas de dichos archivos son consideradas datos no estructurados ya que no podemos realizar ningún tipo de búsqueda sobre ellas. Hoy en día, no podemos realizar consultas dentro de las cadenas de bits de un archivo de audio MP3 buscando patrones para quizá encontrar que dicho archivo contiene una canción, y que la voz dentro de dicha canción quizá es la de Elvis Presley.

Las etiquetas ID3 en este caso serían el ejemplo de los datos semiestructurados. Esto ya que cada uno de los archivos puede o no contener todas o algunas de las etiquetas. Dichas etiquetas nos ayudan a identificar atributos internos de cada uno de los archivos y de esa manera darle una estructura y un significada a esos datos y convertirlos en información. Otro ejemplo de los datos semiestructurados dentro de esta tesis, es el índice dentro del cual se realizan las búsquedas (el cual será explicado a detalle en el **Capítulo III**). Dicho índice fue formulado utilizando el lenguaje de marcado *XML*, en el cual dicho documento puede o no contener todos los atributos que describen a los archivos de audio y eso no afecta el funcionamiento del mismo.

Finalmente nos encontramos con los datos estructurados que en esta tesis están representados por una base de datos relacional dentro de la cual se almacena toda la información de cada uno de los archivos de audio que se encuentren en el sistema. Dicha base de datos, contiene una estructura rígida y constante la cual nos ayuda a darle una semántica completa a los datos que en ella se encuentran y de esa manera transformar dichos datos en información.

2.3.3 Índices y Metadatos

Es indiferente realmente con qué tipo de datos contemos, ya sean estructurados, no estructurados o semiestructurados, lo que realmente interesa es poder obtener dichos datos y transformarlos en información. Pero para poder realizar dicha tarea debemos primero contestar dos preguntas de suma importancia: ¿sobre qué voy a buscar? y ¿dónde voy a buscar? Son las respuestas a estas dos preguntas las que definen un motor de búsqueda de alta o baja calidad. Lo primordial aquí es obtener información, no solo datos, sino información; y no solo información, sino información que en realidad nos vaya a ser de utilidad. Esta es la verdadera dificultad al construir un sistema de búsquedas.

Cada tipo de dato cuenta con ciertos atributos los cuales hacen único a dicho dato. Por ejemplo, los archivos de audio cuentan con atributos como: título, autor, año de creación, género, etc. Todos estos atributos hacen que cada uno de los archivos de audio sean únicos. Estos son los elementos o atributos sobre los cuales haremos las búsquedas. El problema es que dichos atributos de los archivos de audio no se encuentran internamente codificados dentro del archivo. Excepto por el nombre del archivo, no hay nada dentro del archivo que nos indique el valor de cada uno de los atributos que definen a un archivo de audio. Esto es un gran problema porque entonces realmente no podemos realizar búsquedas sobre ningún atributo y entonces nuestro modelo falla. Claro, podríamos internarnos en las cadenas de bits del archivo a buscar patrones que definan que dicho archivo de audio contiene una canción interpretada por *Pink Floyd* por ejemplo. Esa exactamente es la problemática por resolver en el área de reconocimiento de voz, pero hasta hoy, todavía no se han encontrado algoritmos eficientes que resuelvan dicho problema satisfactoriamente. Quizá algún día se logre, pero por lo pronto, necesitamos ayudarnos de otras técnicas las cuales nos ayuden a definir e identificar cada uno de los atributos presentes en los diferentes tipos de archivos o datos.

Una de las técnicas más eficientes para resolver este problema, es la utilización de los llamados *metadatos*. Los metadatos, en realidad, simplemente son datos acerca de los datos. Son una definición o una descripción de los datos. Los metadatos nos ayudan

a darle contexto a los datos y nos facilitan la tarea de realizar búsquedas para obtener información de dichos datos. Los metadatos principalmente nos ayudan a etiquetar o a definir ciertos atributos de algún archivo o tipo de dato en particular. Los ejemplos más claros, en computación, de metadatos son los archivos *HTML* y los archivos *XML*. Como lo mencionamos anteriormente, estos dos tipos de archivos pertenecen al conjunto de los datos semiestructurados pero a su vez, son un lenguaje de metadatos los cuales nos ayudan a definir los atributos de casi cualquier tipo de dato o archivo que deseemos. Los archivos MP3 utilizan esta técnica en las etiquetas ID3 mencionadas y explicadas anteriormente. Realmente lo que estas etiquetas hacen es definir claramente los atributos de cada uno de los archivos. Los lenguajes *HTML* y *XML* realizan exactamente la misma tarea. Cada una de las etiquetas en *HTML*, describe una representación gráfica de un cierto documento de texto dentro de un navegador web. Tomemos como ejemplo el siguiente fragmento de código *HTML*:

```
<title>wAudio - audio portal</title>
```

La etiqueta `<title>` define que el texto "`wAudio - audio portal`", el cual se encuentra dentro de ella, se refiere al título del documento. Si dicha etiqueta no especificara que el texto contenido dentro de ella se refiere al título del documento, el navegador web desplegaría dicho texto tal y como está y no lo desplegaría como el título del documento. Esta técnica de etiquetado utiliza datos semiestructurados para definir una semiestructura de los datos no estructurados.

De igual manera el lenguaje *XML* utiliza las etiquetas para definir ciertos atributos de algún documento o archivo en particular. Tomemos por ejemplo el siguiente texto:

```
Norman J. Rodea Merino 1980 Bosques de Sauce #291 México DF
55967354 5521890355 5000
```

El texto anterior, realmente carece de sentido ya que de tomarlo así, sin ninguna estructura realmente no nos dice nada. No es información. Podemos ayudarnos de los metadatos, en este caso del lenguaje *XML*, para definir los atributos referentes al texto anterior:

```
<nombres>Norman J.</nombres>
<apellido paterno>Rodea</apellido paterno>
<apellido materno>Merino</apellido materno>
<fecha de nacimiento>1980</fecha de nacimiento>
<direccion>Bosques de Sauce #291 México DF</direccion>
<telefono>55967354</telefono>
<movil>5521890355</movil>
<salario>5000</salario>
```

Utilizando la técnica de etiquetado con metadatos, la definición del texto anterior ahora tiene mucho más sentido. Hemos definido atributos como el apellido paterno, la dirección, el salario, etc. De esta manera los atributos característicos de dicho texto han quedado claramente definidos y ahora sí es posible realizar algún tipo de búsqueda sobre el texto para obtener así la información deseada. Entonces en realidad podemos concluir que las búsquedas que se realizan sobre este tipo de datos, los cuales se encuentran etiquetados, no se efectúan sobre los datos en sí, sino sobre las etiquetas que definen los atributos de dichos datos. Esto es muy útil ya que el sistema de búsquedas queda realmente definido y restringido a ciertos atributos en particular sobre los cuales se realizan las búsquedas. El sistema busca sobre las etiquetas y no sobre el texto en sí, esto reduce el tiempo de búsqueda ya que el sistema no tiene que recorrer el documento o el archivo completo, sino solamente enfocarse a encontrar la etiqueta deseada y olvidarse del resto.

Esta técnica es ampliamente utilizada por muchos motores de búsquedas. El ejemplo más característico de un sistema que utilice ésta técnica, es el buscador de páginas web *Google*. El robot de búsquedas de *Google*, el cual recorre toda la red en busca de documentos *HTML* para agregarlos a su índice, realiza las búsquedas sobre las etiquetas del documento *HTML* y no sobre el texto contenido dentro de dicho documento. Se realiza de esta manera, para poder asignarle un grado de importancia distinto a cada una de las diferentes palabras que se encuentran dentro del documento, y para filtrar ciertas palabras o etiquetas las cuales no son de ninguna importancia relevante para el sistema.

Es evidente que no todas las palabras contenidas dentro de un documento cuentan con el mismo grado de importancia. Por ejemplo, las palabras que se encuentren dentro de la

etiqueta que define un tamaño de tipografía mayor, recibirá un grado de importancia mayor que el resto de las palabras. Las palabras contenidas dentro de la etiqueta que define el título del documento también reciben un grado de importancia mayor que el resto de las palabras contenidas en el documento.

Una vez definidos y etiquetados los atributos sobre los cuales se realizarán las búsquedas, la primer pregunta que nos hacíamos anteriormente (¿sobre qué voy a buscar?), ha quedado contestada. Ahora debo de definir un espacio o un lugar común en donde se realizarán las búsquedas. Esto puede ser desde una base de datos hasta una colección de archivos etiquetados (*XML*) sobre los cuales se realizan las búsquedas. Recordemos que en este trabajo de tesis utilizamos archivos de metadatos *XML* para definir y delimitar los atributos sobre los cuales se harán las búsquedas de los archivos de audio MP3.

Para este espacio, necesitamos algún elemento que no sea tan pesado en memoria para poder tenerlo cargado en memoria todo el tiempo. Evidentemente podríamos utilizar la base de datos, pero recordemos que los datos contenidos dentro de ella son considerados datos estructurados los cuales cuentan con una estructura rígida y estable. ¿Qué sucedería si alguno de los archivos MP3 no contara con toda la información requerida por la base de datos?, quizá nuestro sistema fallaría. Necesitamos una estructura mucho más flexible la cual nos permita tener o no tener los atributos definidos. Para eso, nos hemos ayudado de los datos semiestructurados, en especial del lenguaje *XML*. El problema ahora es que por cada archivo de audio MP3 contenido en el sistema, tendremos un archivo *XML* dentro del mismo. Sería muy complicado tener cargados en memoria principal cada uno de los documentos *XML;* y además, ocuparían demasiado espacio en memoria. Por lo contrario, si mantenemos todos los documentos *XML* en el disco duro, el sistema de búsquedas tendría que obtener dichos documentos de la memoria secundaria y cargarlos a memoria principal, lo cual tomaría un tiempo de ejecución muy largo. Lo que necesitamos es encontrar la manera de cargar todos los documentos *XML* a memoria principal para su rápido acceso, pero sin que ocupen toda la memoria de la misma. Necesitamos también algo que nos permita realizar búsquedas sencillas y rápidas sin tener que recorrer el contenido completo de cada uno de los documentos generados a partir de los archivos de audio contenidos en el sistema. La solución a este problema es la generación de un índice dinámico sobre el cual se realicen las búsquedas; y que dicho índice no sea muy pesado en memoria de manera de

que pueda ser cargado completamente en memoria principal y de esta manera ahorrarnos el tiempo que toma obtener un archivo o un documento de la memoria secundaria, en este caso, el disco duro.

Quizá Ockham tenía razón al decir que de todas las soluciones o explicaciones posibles que se presenten a un problema, la más sencilla suele ser la mejor. Lo que se desea es optimizar el sistema de búsquedas y la recuperación de la información; si hoy en día existen técnicas que se han utilizado desde hace mucho tiempo y éstas funcionan eficientemente, entonces hagamos uso de estas herramientas.
El manejo de índices no es nada nuevo, lo podemos ver presente casi en cualquier parte. Cuando uno busca información dentro de un libro, el primer lugar dónde uno busca es en el índice del libro. Los sistemas operativos utilizan índices para organizar la repartición de memoria. La gran mayoría de los sistemas de gestión de bases de datos utilizan índices para ordenar y disminuir la cantidad de información sobre la que se realizan las búsquedas. No es fin de éste trabajo de tesis reinventar el “hilo negro”, simplemente utilizar una seria de técnicas y herramientas existentes para optimizar un sistema de búsquedas.

Para la creación de dicho índice se juntaron cada uno de los documentos *XML*, representativos de los atributos de los archivos de audio contenidos en el sistema, dentro de una colección de documentos *XML*. Esta colección es cargada en memoria principal con la ayuda de un software llamado *eXist* (http://exist.sourceforge.net). *eXist* es un software tipo *open source* (código abierto), el cual realiza la función de gestión de bases de datos nativas en *XML*, es decir, no administra una base de datos relacional, sino más bien una base de datos de colecciones de documentos *XML*. Es un modelo lógico para documentos *XML* el cual almacena y recupera documentos de acuerdo a dicho modelo. De manera que podemos cargar la colección completa de documentos *XML* en memoria principal y *eXist* se encarga de administrarla sin tener que perder el modelo de metadatos sobre los que se realizan las búsquedas. Evidentemente esta herramienta nos da la facilidad de agregar documentos a la colección en cualquier momento, de manera que obtenemos un índice dinámico de documentos *XML* (metadatos) sobre los cuales realizamos las búsquedas; y la segunda pregunta que nos hacíamos anteriormente (¿dónde voy a buscar?), queda contestada.

2.3.4 Estrategias de Recuperación de Información

En los apartados anteriores, se describieron las distintas técnicas que el sistema, descrito dentro de éste documento de tesis, utiliza para realizar las búsquedas sobre los archivos de audio MP3. Ahora, debemos encontrar la manera de calificar dichos resultados con respecto a la relevancia que tienen con la consulta que los generó, es decir, debemos asignarle un valor de importancia a cada uno de los resultados generados de manera que los podamos ordenar con respecto a su relevancia con respecto a la consulta. Este proceso es conocido en inglés comúnmente como *ranking*. El propósito es no solamente obtener resultados de una consulta, sino ordenar dichos resultados en grado de importancia hacia la consulta, de manera que los primero resultados presentados al usuario sean los más relevantes a su consulta.

Existen varios modelos los cuales nos ayudan a definir una estrategia de calificación de resultados eficiente. A continuación presentamos algunos de los más conocidos: (Tomamos por hecho para todas las siguientes definiciones, que contamos con una consulta introducida por el usuario y una colección de documentos sobre la que se realizarán las búsquedas y se aplicarán los distintos modelos.)

- **Modelo de Espacios Vectoriales**

  La consulta y cada documento dentro de la colección son representados como vectores dentro de un modelo espacial, y se mide el grado de similitud entre ambos vectores. Es el modelo más popular hoy en día ya que permite discriminar correctamente entre documentos.

- **Modelo Probabilística**

  Para cada término dentro de la colección, se calcula la probabilidad de que dicho término aparezca dentro de cada uno de los documentos. Para los términos que coinciden en la consulta y en un documento, el grado de relevancia es obtenido al combinar las probabilidades de cada uno de los términos previamente calculadas.

- **Redes de Inferencia**

  Se utiliza una red Bayesiana para inferir la relevancia de un documento hacia la consulta. El grado de inferencia obtenido, es utilizado como un coeficiente de similitud entre el documento y la consulta.

- **Indexación Boleana**

  Se asocia un peso a cada término de la consulta de manera que ése peso es utilizado para calcular el coeficiente de similitud.

- **Redes Neuronales**

  Son una secuencia de “neuronas” o nodos en la red los cuales son activados cuando una consulta activa las ligas a los distintos documentos. La fuerza de cada una de las ligas en la red es transmitida al documento y recuperada para formar un coeficiente de similitud entre la consulta y el documento. Las redes neuronales son entrenadas o calibradas ajustando los pesos de las ligas en respuesta de documentos predeterminados relevantes o irrelevantes.

- **Algoritmos Genéticos**

  Una consulta óptima para encontrar documentos relevantes puede ser generada por evolución de genes. Las consultas son representadas por genes. Una consulta inicial es utilizada con pesos aleatorios o estimados. Nuevas consultas son generadas modificando esos pesos. Una consulta nueva sobrevive acercándose a documentos relevantes conocidos, y las consultas que menos se acerquen son eliminadas.

Existen más modelos de recuperación de información pero consideramos que los mencionados anteriormente son los más utilizados ya que son los que generan los mejores resultados.

Como nos pudimos dar cuenta, ya no es suficiente definir si cierto término o palabra se encuentra dentro de algún documento o no. Es necesario definir que tan relevante es dicha palabra dentro de cada uno de los documentos. Antiguamente, nos bastaba saber si la palabra que buscamos se encuentra en el documento o no; una simple respuesta boleana: si o no. El problema hoy en día, es que la cantidad de documentos en

los que podemos buscar crece exponencialmente. Solamente en Internet se tienen calculadas aproximadamente 8,000,000,000 de páginas web hoy en día. Ya no es como antes que realizábamos búsquedas dentro de la colección de una sola biblioteca. Hoy en día gracias al avance en las bibliotecas digitales, podemos realizar búsquedas dentro de cientos de bibliotecas interconectadas alrededor del mundo, las cuales cuentan cada una con miles de ejemplares sobre los cuales buscar. Entonces es realmente importante definir una buena técnica de recuperación de información la cual no solamente se enfoque en respuestas boleanas, sino más bien en frecuencias y pesos asignados a palabras relevantes a ciertos documentos; y calcular un grado de similitud o relevancia entre dichos documentos y las consultas insertadas.

Pero entonces que sucede cuando las palabras o términos contenidas dentro de los documentos por buscar no se repiten, sino que solamente son mencionados una vez en cada documento. Evidentemente puede que dichos términos se repitan en varios documentos, pero que sucede si dentro del documento solamente son mencionadas una sola vez. Es importante mencionar este contraejemplo, ya que la gran mayoría de los modelos de recuperación de información se basan en las frecuencias de repetición de los términos dentro de los documentos. Tomemos el modelo de espacios vectoriales por ejemplo. Dicho modelo obtiene un valor que representa la cantidad de veces que una palabra se repite en un documento, a ese valor se le conoce como frecuencia. Después, se calcula la frecuencia de esa misma palabra contra el resto de la colección, es decir, obtiene la frecuencia de la palabra en cada documento y se calcula la sumatoria para obtener una frecuencia total de dicha palabra en la colección. Tal frecuencia es utilizada para obtener el grado de relevancia o similitud al momento que se ingresa una consulta. Mientras la frecuencia de la palabra dentro de un documento sea mayor, mayor será su grado de relevancia. Indirectamente, mientras mayor sea la frecuencia de dicha palabra dentro de la colección, menor será el grado de relevancia. Es decir, si una palabra se repite muchas veces dentro de un documento, pero pocas veces en toda la colección, dicha palabra tendrá un alto grado de relevancia para la consulta. Si una palabra se repite pocas veces dentro de un documento, pero muchas veces en toda la colección, dicha palabra tendrá un bajo grado de relevancia para la consulta.

La razón por la que se menciona este contraejemplo es la siguiente. Tomemos en cuenta los documentos sobre los cuales se realizan las búsquedas en el sistema descrito

por éste documento de tesis. Los documentos sobre los que se realizan las búsquedas, son documentos *XML* los cuales describen los atributos de cada uno de los archivos de audio MP3 dentro del sistema. Cada documento cuenta con los atributos descritos por el sistema (los cuales se explicarán a detalle en el **Capítulo III**) los cuales definen cada archivo de audio. Algunos de estos atributos son: el título de la canción, el artista que la interpreta, y el álbum al que pertenece. Pongamos de ejemplo que un cierto documento *XML* describe un archivo que contiene una canción llamada *Time* interpretada por el grupo *Pink Floyd* la cual es parte del álbum *Dark Side Of The Moon*. Entonces nos podemos dar cuenta de que dentro del documento que describe dicho archivo, solamente son mencionadas una vez cada palabra. Es decir, la palabra *Floyd* no se volverá a repetir en todo el documento, igual y se llega a repetir en algún otro documento dentro de la colección, pero no dentro de dicho documento. Entonces la frecuencia de cada uno de los términos o palabras contenidos dentro de este documento siempre será la misma. De la misma manera, la frecuencia de la palabra *Floyd* en cada uno de los documentos que sea mencionada, será la misma que en todos los documentos. De tal manera que si tomamos en cuenta todos los documentos los cuales contienen la palabra *Floyd*, todos presentarán la misma frecuencia para dicha palabra y entonces realmente no obtendremos un grado de relevancia distintivo con el cual podamos ordenar los resultados de alguna consulta que contenga la palabra *Floyd*.

Esto aún sin tomar en cuenta de que la consulta se realiza sobre las etiquetas y no sobre los datos contenidos dentro de los documentos, o sea, las búsquedas se realizan sobre las etiquetas que definen al título de la canción, al artista, al álbum, entre otras; de manera de que raramente obtendremos un documento en el cual alguna palabra se repita dentro de la misma etiqueta. Esto nos crea un gran problema ya que entonces realmente ninguno de los modelos mencionados anteriormente nos sirve. Dichos modelos fueron diseñados para realizar búsquedas sobre documentos los cuales contienen gran cantidad de texto como lo son: libros, páginas web, ensayos, publicaciones académicas, revistas, periódicos, etc. Pero en realidad no son tan eficientes cuando nos referimos a documentos los cuales cuentan con muy poco texto y sus atributos se encuentran perfectamente bien definidas. Los modelos anteriormente mencionados fueron diseñados principalmente para realizar un tipo de búsqueda llamado en inglés *Full Text* (a texto abierto). No estamos concluyendo que dichos modelos no nos vayan a servir, simplemente se está concluyendo que necesitamos adaptar tales modelos al tipo de búsquedas que nuestro sistema realiza. Necesitamos diseñar un modelo de recuperación

de información el cual esté basado en las ideas principales de los modelos anteriormente mencionados, pero que resuelva la problemática que esta situación presenta.