Introducción

HTML (HyperText Markup Language) es un lenguaje muy sencillo que permite describir hipertexto, es decir, texto presentado de forma estructurada y agradable, con enlaces (hyperlinks) que conducen a otros documentos o fuentes de información relacionadas, y con inserciones multimedia (gráficos, sonido...) La descripción se basa en especificar en el texto la estructura lógica del contenido (títulos, párrafos de texto normal, enumeraciones, definiciones, citas, etc) así como los diferentes efectos que se quieren dar (especificar los lugares del documento donde se debe poner cursiva, negrita, o un gráfico determinado) y dejar que luego la presentación final de dicho hipertexto se realice por un programa especializado (como Mosaic, o Netscape).

El lenguaje HTML es un texto enriquecido con elementos multimedia interpretado por el navegador del usuario, dando como resultado una forma mas explicita y fácil de comunicación entre las computadoras.

Para enriquecer mas una página WEB se cuenta con los Applets y Java, Los applets son pequeños programas ejecutables escritos en lenguaje Java, que se pueden colocar en nuestro servidor, junto con el resto de ficheros que componen un sitio Web (documentos HTML, ficheros de imagen, sonido, etc.) para lograr múltiples efectos con texto, imágenes, sonidos, etc.

Java es un lenguaje de programación orientado a objetos que ha sido creado y desarrollado por la compañía Sun Microsystems. Es independiente de cualquier plataforma y puede ejecutarse en cualquier ordenador que tenga un navegador compatible con Java (Netscape 2.x o superior, Explorer 3.0 o superior, o HotJava).

Hoy día el web, es una puerta abierta hacia una nueva experiencia multimedia, donde se pueden ver programas de televisión, crear audio-video conferencias con nuestros amigos o familiares, escuchar clips antes de comprar nuestro cd, escuchar estaciones de radio en vivo, estar presentes en una clase remota, o tal vez presenciar eventos en vivo. Sin embargo, otras soluciones como el Java Media Framework, están rápidamente avanzando en este campo, y sobre todo, porque ofrecen un recurso gratuito al alcance de todo mundo. Lo único que se necesita es la capacidad de cargar por lo menos 200 kbps, éste es el ancho de banda mínimo que se requeriría aunque sean pocas conexiones simultaneas.

El gran impulso que actualmente tienen los elementos multimedia en el web, es gracias al desarrollo que ha tenido el área de las comunicaciones. Cada vez, se cuenta con una velocidad de transferencia mucho más rápida, a tal grado que en no mucho tiempo, podamos recibir audio y video, con una resolución superior a un televisor convencional.

1. Clientes Estáticos

1.1 Introducción al HTML

HTML Hyper Text Markup Language

El HTML, Hyper Text Markup Language (Lenguaje de marcación de Hipertexto) es el lenguaje de marcas de texto utilizado normalmente en la www (World Wide Web). Fue creado en 1986 por el físico nuclear Tim Berners-Lee; el cual tomó dos herramientas preexistentes: El concepto de Hipertexto el cual permite conectar dos elementos entre si y el SGML (Lenguaje Estándar de Marcación General) el cual sirve para colocar etiquetas o marcas en un texto que indique como debe verse. HTML no es propiamente un lenguaje de programación como C++, Visual Basic, Cobol etc., sino un sistema de etiquetas. HTML no presenta ningún compilador, por lo tanto algún error de sintaxis que se presente éste no lo detectará y se visualizara en la forma como éste lo entienda.

El entorno para trabajar HTML es simplemente un procesador de texto, como el que ofrecen los sistemas operativos Windows (Bloc de notas), UNIX (el editor vi o ed) o el que ofrece MS Office (Word). El conjunto de etiquetas que se creen, se debe guardar con la extensión .htm o .html.

Estos documentos pueden ser mostrados por los visores o “browsers” de paginas Web en Internet, como Netscape Navigator, Mosaic, Opera y Microsoft Internet Explorer.

También existe el HTML Dinámico (DHTML), que permite crear efectos especiales.

1.2 Orígenes del HTML

En 1986, la publicación de la ISO 8879 presenta el Standard General Markup Language, origen del HTML.

En 1989, Tim Berners-Lee, en el Centro Europeo de Investigaciones Nucleares presenta su artículo Information Management: dedicándose de lleno al desarrollo de un sistema que permitiera el acceso en línea de manera uniforme a la información

disponible en muchos recursos distintos, y que pudiese funcionar en máquinas que sean conectadas por redes basadas en TCP/IP.

Entre 1990 y 1991. Tim Berners-Lee define el HTML como un subconjunto de SGML (Standard Generalized Markup Language), que más tarde se llamará nivel 0; soporta encabezados, listas y enlaces. Se crea el nombre World Wide Web.

En 1991. Tim Berners-Lee introduce el primer visor de HTML, LineMode, que trabaja en modo texto y sólo en plataformas UNIX. El Centro Europeo de Investigaciones Nucleares realiza la apertura del primer sitio con acceso público de World Wide Web el 17 de mayo.

En 1992. Dan Connolly produce la primera Definición de Tipo de Documento (DTD) para el lenguaje, llamada HTML 1.0, agregando a la definición original atributos para modificar el estilo físico del texto. Se distribuye Viola, primer visor gráfico de Web y disponible sólo para X.11.

En 1993. Un nuevo visor que soporta un mayor nivel, Lynx, es producido por la Universidad de Kansas, si bien lee sólo texto. Aparece Mosaic, desarrollado por el Centro Nacional para Aplicaciones de Supercomputadoras, es el primer visor de Web en entorno gráfico que se hace disponible para computadoras personales, lo que lo hace inmediatamente popular. A fines de año, comienzan a aparecer los primeros artículos sobre WWW en diarios y revistas de circulación masiva. Tim Berners-Lee utiliza el trabajo del año anterior de Connolly para presentar el borrador de la primera norma (RFC -Recommendation for Comments) de HTML para Internet.

En 1994. La Universidad Técnica de Graz desarrolla un servidor y clientes con mayores prestaciones para HTML, Hyper-G, que no tiene gran éxito. Cello, primer visor de HTML que no requiere TCP/IP presentado por la Escuela de Leyes de la Universidad de Cornell. Dan Connolly y Karen Olson Muldrow redefinen el HTML para el nivel 2.0, que ahora soporta formularios. Un grupo de programadores que desarrollaran el Mosaic producen un nuevo visor de World Wide Web, Netscape (también conocido como Mozilla), que tiene una amplia aceptación entre los usuarios, pero que soporta elementos de programación que equivalen a una degeneración del HTML (tamaños de letra, fondos). Se define un equivalente para los modelos en tres dimensiones del HTML, el VRML (Virtual Reality Modeling Language), que permite moverse dentro de los ambientes definidos. En este mismo año se realizan la Primera y Segunda conferencias internacionales de WWW, en Ginebra y Chicago, respectivamente. Se crea la W3 Organization.

En 1995. Dave S. Raggett (Hewlett-Packard, Inglaterra) comienza a compilar la normativa del nuevo nivel del lenguaje, el HTML 3.0, cuya principal novedad es el soporte de tablas. Microsoft produce su primer visor de Internet, el cual también utiliza elementos de HTML degenerados. Una nueva versión de Netscape, Navigator 2.0, agrega soporte de encuadres. Sun Microsystems produce el primer visor de World Wide Web con soporte de un lenguaje de programación, HotJava. Se celebran la Tercera y Cuarta conferencias internacionales de WWW, en Boston y Darmstadt respectivamente, y la conferencia de WWW para Asia y el Pacífico en Wagga-Wagga.

En el año 1996. Netscape Communications y Microsoft presentan las nuevas versiones de sus visores que soportan gran parte del nivel de HTML 3.0. Aparecen visores no comerciales que implementan la norma completa de HTML 3.0. Se formaliza un nuevo nivel para la modelación en tres dimensiones, VRML 3.0, que permite interactuar con los objetos definidos. Se celebra la Quinta conferencia internacional de WWW en Rocquencourt.

En 1997. D. Raggett presenta, en enero, la versión normalizada del 3.2. En julio, aparece la versión 4.0, experimental.

En 1998, apareció HTML 4.0 y después de ello se ha polularizado la utización de wizards para crear nuestras propias páginas web, sin la necesidad de saber el código que viene detrás.

1.3 Creación de páginas web con lenguaje HTML

Para crear una página web se pueden utilizar varios programas especializados en esto, como por ejemplo, el Microsoft Front Page o el Macromedia Dreamweaver 3. Otra forma de diseñar un archivo .html, es copiar todo en el Bloc de Notas del Windows, ya que este sencillo programa cumple con un requisito mínimo que es la posibilidad de trabajar con las etiquetas con las que trabaja este lenguaje.

Estructura Básica de un documento HTML

El principio esencial del lenguaje HTML es el uso de las etiquetas (tags). Funcionan de la siguiente manera:

<XXX> Este es el inicio de una etiqueta.
</XXX> Este es el cierre de una etiqueta.

Las letras de la etiqueta pueden estar en mayúsculas o minúsculas, indiferentemente.

Lo que haya entre ambas etiquetas estará influenciada por ellas. Por ejemplo, todo el documento HTML debe estar entre las etiquetas <HTML> y </HTML>:

<HTML> [Todo el documento] </HTML>

Un documento HTML en sí está dividido en dos zonas principales:

El encabezamiento, comprendido entre las etiquetas <HEAD> y </HEAD>
El cuerpo, comprendido entre las etiquetas <BODY> y </BODY>

Dentro del encabezamiento hay información del documento, que no se ve en la pantalla principal del BROWSER que es utilizado para visualizar el documento HTML, principalmente la información encontrada en el encabezamiento es el título del documento, comprendido entre las etiquetas <TITLE> y </TITLE>. El título debe ser breve y descriptivo de su contenido, pues será lo que vean los demás cuando añadan nuestra página a su bookmark (o agenda de direcciones).

Dentro del cuerpo está todo lo que se quiere que aparezca en la pantalla principal (texto, imágenes, etc.)

Por tanto, la estructura de un documento HTML queda de esta manera:

```
<HTML>
<HEAD>
<TITLE> Título de la página </TITLE>
</HEAD>
<BODY>
[Aquí van las etiquetas que se visualiza en la página]
</BODY>
</HTML>
```

1.4 Dando forma al texto del documento HTML.

Cuando se escribe en el documento el texto que se quiere que aparezca en la pantalla, se puede observar que éste se acomoda a ella, sin que se tenga que pulsar el retorno del carro. Si se necesita separar el texto en distintos párrafos se puede usar la etiqueta <P>, (que no tiene su correspondiente etiqueta de cierre </P>)

El texto puede tener unas cabeceras, comprendidas entre las etiquetas <H1> y </H1>, <H2> y </H2>, etc. (hasta el número 6), siendo el número indicativo del tamaño. El tamaño mayor es el correspondiente al número 1.

Una etiqueta muy interesante es la de centrado <CENTER> y </CENTER> (no la soportan todos los navegadores, aunque sí la mayoría de ellos). Nos centra todo lo que esté dentro de ella, ya sea texto, imágenes, etc.

También se cuenta con los separadores (horizontal rules), que se consiguen con la etiqueta <HR> (no existe la correspondiente de cierre). Con ella se obtiene una raya horizontal tan ancha como la pantalla, y con la apariencia de estar embutida sobre el fondo, (Ver Anexos-Ejemplo 1).

Una etiqueta puede estar anidada dentro de otra. En el ejemplo anterior cómo lo está la etiqueta <CENTER> dentro de la etiqueta <H1>.

Cuando se desea poner un texto sin ninguna característica especial, se lo coloca directamente. Únicamente, la separación entre párrafos (dejando una línea en blanco) se lo consigue con la etiqueta <P>.

Si se desea separar los párrafos, o cualquier otra cosa, pero sin dejar una línea en blanco, se utiliza una etiqueta parecida
 (break, o romper). Tampoco tiene etiqueta de cierre.

Al escribir el texto, si se coloca más de un espacio en blanco entre dos palabras se observa que el navegador sólo reconoce uno de ellos. Si se pretende forzar a que lo haga, se coloca el código " " (non-breaking space).

Para destacar alguna parte del texto se pueden usar:
<B> y </B> para poner algo en negrita (bold).
<I> y </I> para poner algo en cursiva (italic).

Otra etiqueta interesante es <PRE> y </PRE>. El texto que se encuentre entre ella estará preformateado, es decir que aparecerá como si hubiera sido escrito con una máquina de escribir, con una fuente de espaciado fijo (tipo Courier). Además se respetarán los espacios en blanco y retornos del carro, tal como estaban en nuestro documento HTML. Es muy apropiada para confeccionar tablas y otros documentos similares.

Con la etiqueta <TT> y </TT> se consigue también que el texto tenga un tamaño menor y la apariencia de los caracteres de una máquina de escribir (typewriter). La diferencia con la anterior es que no preformatea el texto, sino que únicamente cambia su apariencia.

En las fórmulas matemáticas puede interesar poder escribir índices y subíndices, que se consiguen con las etiquetas y respectivamente.

A menudo nos interesará presentar las cosas en forma de listas. Se tiene que escoger entre tres tipos distintos:

Listas desordenadas (no numeradas)
Listas ordenadas (numeradas)
Listas de definición.

Las listas desordenadas (Unordered Lists) sirven para presentar cosas que, por no tener un orden determinado, no necesitan ir precedidas por un número. Su estructura es la siguiente:

```
<UL>
<LI> Un elemento
<LI> Otro elemento
<LI> Otro elemento más
<LI> etc.
</UL>
```

Es decir, toda la lista está dentro de la etiqueta <UL> y </UL>, y luego cada elemento va precedida de la etiqueta <LI> (list ítem). El resultado de lo anterior es el siguiente:

Se puede anidar una lista dentro de otra. Por ejemplo:

```
<UL>
<LI> Mamíferos
<LI> Peces
  <UL>
  <LI> Sardina
  <LI> Bacalao
  </UL>
<LI> Aves
</UL>
```

Las listas ordenadas (Ordered Lists) sirven para presentar elementos en un orden determinado. Su estructura es muy similar a la anterior. La diferencia estriba en que en el resultado aparecerá automáticamente un número correlativo para cada elemento.

<OL>
<LI> Primer Elemento
<LI> Segundo Elemento
<LI> Tercer Elemento
<LI> etc.
</OL>

Al igual que las listas desordenadas, también se pueden anidar las listas ordenadas.

El tercer tipo lo forman las listas de definición. Como su nombre indica, son apropiadas para glosarios (o definiciones de términos). Toda la lista debe ir englobada entre las etiquetas <DL> y </DL>. Y a diferencia de las dos que se ha visto, cada renglón de la lista tiene dos partes:

El nombre de la cosa a definir, que se consigue con la etiqueta <DT> (definition term).
La definición de dicha cosa, que se consigue con la etiqueta <DD> (definition definition).

<DL>
<DT> Una cosa a definir
<DD> La definición de esta cosa
<DT> Otra cosa a definir
<DD> La definición de esta otra cosa
</DL>

Comentarios no visibles en la pantalla

A veces es muy útil escribir comentarios en el documento HTML sobre el código que se escribe, que nos pueden servir para recordar posteriormente sobre lo que se hizo, y que no se quiere ver en pantalla.

Esto se consigue encerrando dichos comentarios entre estos dos símbolos: <!-- y --> (Ver Anexos-ejemplo 2).

1.5 Enlaces

La característica que más ha influido en el espectacular éxito del WEB ha sido, aparte la de su carácter multimedia, la posibilidad de unir los distintos documentos repartidos por todo el mundo por medio de enlaces hipertexto.

En general, los enlaces tienen la siguiente estructura:
<A HREF="XXX"> YYY </A>

Donde XXX es el destino del enlace (Obsérvese las comillas). YYY es el texto indicativo en la pantalla del enlace (con un color especial y generalmente subrayado)

Tipos de enlaces

Enlaces dentro de la misma página
Enlaces con otra página nuestra
Enlaces con una página fuera de nuestro sistema
Enlaces con una dirección de e-mail

1.5.1 Enlaces dentro de la misma página

A veces, en el caso de documentos (o páginas) muy extensos, puede interesar dar un salto desde una posición a otra determinada. En este caso, lo que antes se llama XXX, es decir, el destino del enlace, en este caso el sitio dentro de la página a donde se quiere saltar, se sustituye por #MARCA (la palabra MARCA puede ser cualquier palabra). Lo que se llama antes YYY es la palabra (o palabras) que aparecerán en la pantalla en color (en forma de hipertexto). Su estructura es, entonces:

<A HREF="#MARCA"> YYY </A>

Y en el sitio exacto a donde se quiere ubicar, se debe poner la siguiente etiqueta:
<A NAME="MARCA"> </A>

1.5.2 Enlaces con otra página nuestra

Puede ser que se tenga una sola página. Pero lo más frecuente es que se tenga varias páginas, una inicial (o principal) y otras conectadas a ella, e incluso entre ellas mismas.

Si se quiere enlazar con la página creada en el ejemplo anterior, que se la ha llamado pagpru.html. En este caso, simplemente se substituye lo que se ha llamado XXX (el destino del enlace) por el nombre del archivo:

<A HREF="pagpru.html"> Ejemplo de mi segunda pagina </A>

Si se desea que vaya a un sitio concreto de otra página propia en vez de ir al principio de la página, adonde va por defecto, en ese sitio se tiene que colocar una marca (ver la *Enlaces dentro de la misma página*), y completar el enlace con la referencia a esa marca.

Se lo visualiza con el siguiente ejemplo: <A NAME="MIMARCA"></A> es la marca que se coloca en la pagina, a la que se desea acceder desde otra nuestra. Entonces la etiqueta tiene que ser: <A HREF="pagpru.html#MIMARCA"> En mi otra pagina </A>.

Una observación importante: Pudiera ocurrir que el sitio del WEB en cuestión estuviera organizado con un directorio principal, y otros subdirectorios auxiliares. Si la página a la que se desea acceder está, por ejemplo en el subdirectorio misubdir, entonces en la etiqueta tendría que colocarse "misubdir/pagpru.html".

Y a la inversa, si quiero saltar desde una página a otra que está en un directorio anterior, en la etiqueta tendría que haber puesto "../pagpru.html". Esos dos puntos hacen que se dirija al directorio anterior. Obsérvese que se debe utilizar el símbolo / para indicar los subdirectorios, y no este otro \, que es propio únicamente de Windows.

Se puede tener todo junto en un único directorio, pero esto tiene el inconveniente de que esté todo más desordenado, y sean más difíciles de hacer las futuras modificaciones.

1.5.3 Enlace con una página fuera de nuestra página

Si se desea enlazar con una página que esté fuera de nuestro sistema (es decir, que esté en un servidor distinto al que soporta nuestra página), es necesario conocer su dirección completa, o URL (Uniform Resource Locator). El URL podría ser, además de la dirección de una página del WEB, una dirección de FTP, GOPHER, etc.

Una vez conocida la dirección (o URL), se lo coloca en vez de lo que se ha llamado anteriormente XXX (el destino del enlace). Si se desea enlazar por ejemplo con la página de Netscape (cuyo URL es: http://home.netscape.com/), la etiqueta sería:

<A HREF="http://home.netscape.com/"> Página inicial de Netscape </A>

Es muy importante copiar estas direcciones correctamente (respetando las mayúsculas y minúsculas, pues los servidores UNIX sí las distinguen)

1.5.4 Enlace con una dirección e-mail

En este caso, se substituye lo que se ha llamado antes XXX (el destino del enlace) por mailto: seguido de la dirección de e-mail. La estructura de la etiqueta es:

<A HREF= “mailto: dirección de e-mail”> Texto del enlace </A>

Un ejemplo podría ser:

<A HREF= “mailto: jencalad@uazuay.edu.ec”> Janela Encalada</A>

Hay algunos navegadores que no subrayan el comentario de este tipo de enlace.

Una manera recomendable y más segura para conocer la dirección e-mail seria poner algo así como:

Comentarios a Janela Encaladaa <A HREF="mailto: jencalad@uazuay.edu.ec"> jencalad@uazuay.edu.ec </A>

Es decir, es conveniente, por la razón dicha anteriormente, poner también en el texto del enlace la dirección de e-mail. (Ver Anexos- ejemplo 3)

1.6 Imágenes

La etiqueta que nos sirve para incluir imágenes en nuestras páginas del WEB es muy similar a la de enlaces a otras páginas, que se ha visto anteriormente. La única diferencia es que, en lugar de indicar al programa navegador el nombre y la localización de un documento de texto HTML para que lo cargue, se le indica el nombre y la localización de un archivo que contiene una imagen.

La estructura de la etiqueta es:
<IMG SRC="imagen.gif">

Con el comando IMG SRC (image source, fuente de la imagen) se indica que se quiere cargar una imagen llamada imagen.gif (o el nombre que tenga).

Dentro de la etiqueta se pueden añadir otros comandos, tal como ALT
<IMG SRC="imagen.gif" ALT="descripción">

Con el comando ALT se introduce una descripción (una palabra o una frase breve) indicativa de la imagen. Este comando, que en principio se puede omitir, es en beneficio de los que accedan a nuestra página con un programa navegador en forma de texto como el lynx. Ya que no son capaces de ver la imagen, por lo menos pueden hacerse una idea sobre ella. Pero no es sólo por esto. Hay casos, como se verá más adelante, en los que se utiliza una imagen como enlace a otra página.

Con respecto a la localización del archivo de esa imagen, se puede decir aquí lo mismo que en el capítulo anterior referente a los enlaces. Si no se indica nada especial, como en el caso que se ha expuesto, quiere decir que el archivo imagen.gif está en el mismo directorio que el documento HTML que se esta escribiendo. Si no es así, se siguen los mismos criterios que los indicados para los enlaces.

Las imágenes deben estar guardadas en un formato de archivo especial llamado GIF. (Hay también otro formato más avanzado JPG). Este formato GIF almacena las imágenes con un máximo de 256 colores, en forma comprimida.

Un aspecto muy importante a tener en cuenta es el tamaño de las imágenes, pues una imagen grande supone un archivo grande, y esto puede resultar en un tiempo excesivo de carga, con el consiguiente riesgo de que quien esté intentando cargar la página se canse de esperar, y desista de ello.

Para elegir la posición de la imagen con respecto al texto hay distintas posibilidades. La más sencilla es colocarla entre dos párrafos, con un titular a un lado. Los navegadores más actuales (como el Netscape Navigator y el Microsoft Internet Explorer) permiten que el texto pueda rodear a la imagen.

A continuación se escoge la posición del titular con respecto a la imagen. Se puede poner arriba, en medio o abajo del lado de la imagen. Para ello se añade el comando ALIGN a la etiqueta, de la siguiente manera:

<IMG SRC="logo.gif" ALIGN=TOP>	Alineado arriba
<IMG SRC="logo.gif" ALIGN=MIDDLE>	Alineado en medio
<IMG SRC="logo.gif" ALIGN=BOTTOM>	Alineado abajo

Otra posibilidad muy interesante es la de utilizar una imagen como enlace a otra página. Para estos casos se utilizan generalmente imágenes pequeñas (iconos), aunque se puede usar cualquier tipo de imagen.

Según se revisó anteriormente, la estructura general de un enlace es:

<A HREF="XXX"> YYY </A>

En este caso se substituye XXX por el nombre del archivo de la página a la que se quiere acceder. Y en lugar de YYY se coloca la etiqueta completa de la imagen (que queda así englobada dentro de la etiqueta del enlace).

```
<A HREF="mipagina.html"><IMG SRC="logotipo.gif"></A>
```

Pulsando la imagen se comprueba cómo efectivamente enlaza con la página deseada. Se puede observar además que la imagen está rodeada de un rectángulo del color normal en los enlaces. Si no se desea que aparezca ese rectángulo, hay que incluir dentro de la etiqueta de la imagen el atributo BORDER=0, es decir:

```
<A HREF="mipag.html"><IMG SRC="logotipo.gif" BORDER=0></A>
```

Posicionando el cursor sobre esta última imagen, se comprueba que actúa también como enlace aunque carezca del rectángulo de color. Esto puede resultar más estético, pero se corre el riesgo de que el usuario no se dé cuenta de que la imagen sirve de enlace.

También se puede utilizar una imagen para enlazar con otra imagen. Supóngase que se desea enlazar con la imagen esta imagen.gif por medio de esta otra imagen desde esta.gif :

```
<A HREF="estaimagen.gif"><IMG SRC="desdeesta.gif"></A>
```

Por ultimo, otra posibilidad es la de utilizar un texto para enlazar con una imagen. En este caso se substituye XXX (el destino del enlace) con el nombre del

archivo de la imagen a la que se quiere acceder e YYY (lo que aparece en pantalla como el enlace) por el texto.

Por ejemplo:

<A HREF="isla.gif"> un paraíso tropical </A>

Un tipo de imágenes del que se hace abundante uso y que sirven para mejorar la presentación de la página son los iconos, botones, barras separadoras, etc. A pesar de su tamaño o forma, son imágenes como cualquier otra.

1.6.1 Alineación y dimensionado de imágenes

Alineación de las imágenes

Si se quieren lograr diseños mas llamativos se pueden usar los comandos o atributos conjuntamente con la etiqueta <IMG SRC= “imagen.gif”>, ALIGN=. Donde quiera que se desee que aparezca una imagen basta con insertar:

<IMG SRC="/camino/imagen.gif" ALIGN=LEFT>	Alinea la pagina a la izquierda
<IMG SRC="/camino/imagen.gif" ALIGN=CENTER>	Alinea la pagina al centro
<IMG SRC="/camino/imagen.gif" ALIGN=RIGHT>	Alinea la pagina a la derecha

Si se quiere interrumpir el proceso de rellenado del texto a los lados de la imagen, para que salte hasta debajo de ella, es decir, dejar un espacio en blanco parcialmente, se puede emplear las siguientes extensiones de la etiqueta
:

<BR CLEAR=LEFT>	Busca el primer margen libre (clear) a la izquierda.
<BR CLEAR=RIGHT>	Busca el primer margen libre a la derecha.
<BR CLEAR=ALL>	Busca el primer margen libre a ambos lados.

Un ejemplo para aclarar esto:

<IMG SRC="imagen.gif" ALIGN=LEFT> Este texto esta a un lado de la imagen.

 Este también esta a un lado de la imagen, en la línea siguiente.
<BR CLEAR=LEFT> Este otro texto, en cambio, ha buscado el primer margen libre a la izquierda.

Dimensionando la imagen

Los programas navegadores cuando cargan un documento HTML y encuentran una etiqueta de una imagen, interrumpen el proceso de carga y solicitan al servidor que le envíe dicha imagen, quedando a la espera hasta que se complete el envío, repitiéndose este proceso con cada una de las imágenes.

Esto es especialmente molesto cuando, como ocurre frecuentemente, en la cabecera de la página se encuentra una imagen grande, ya que durante un tiempo relativamente largo no se verá nada en la pantalla.

Para evitar este inconveniente existen unas extensiones de la etiqueta de la imagen <IMG SRC="imagen.gif"> que sirven para indicar al navegador cuáles son sus dimensiones en pixels.

En este caso, el navegador actúa de una forma más favorable, ya que entonces, como conoce las dimensiones de las imágenes les reserva un espacio en la pantalla y va colocando el texto de forma apropiada, sin ninguna interrupción, a la vez que va rellenando esos espacios reservados a las imágenes.

Estos comandos o atributos son WIDTH (ancho) y HEIGHT (alto).

Por ejemplo, para la imagen isla.gif situada más arriba:

```
<IMG SRC="imagen.gif" WIDTH=120 HEIGHT=94>
```

Es conveniente hacer esto con todas las imágenes, incluso con las más pequeñas (iconos, botones, etc.), para que no haya ninguna interrupción en el proceso de carga del documento.

Se puede también, si se quiere, dimensionar las imágenes con unos valores distintos a los que realmente tienen, variando el tamaño, la anchura o la altura. Esto es muy conveniente, por ejemplo para poner en la página un thumbnail (reproducción en pequeño de una imagen), que hace de enlace a la imagen en su verdadero tamaño. De esta manera no se recarga demasiado una página, y el usuario será quien decida qué imágenes desea cargar.

Para hacer que una imagen reducida sea el enlace con la imagen en su tamaño original, se lo consigue con:

<A HREF="imagen.gif"> <IMG SRC="imagen.gif" WIDTH=150 HEIGHT=75> </A>

También se puede conseguir esto de otra manera, más correcta aunque más laboriosa. Es la de reducir en un programa gráfico esta imagen a 150x75, guardarla con otro nombre, y luego hacer que la pequeña sea el enlace de la grande. Es más correcta esta otra solución porque no todos los navegadores reconocen los comandos WIDTH y HEIGHT, incluso tampoco algunas de las versiones más antiguas de Netscape.

1.7 Fondos y colores

Se puede cambiar el fondo de dos maneras distintas:

Con un color uniforme
Con una imagen

Fondos con un color uniforme

Se consigue añadiendo el comando BGCOLOR a la etiqueta <BODY> (situada al principio del documento), de la siguiente manera:
<BODY BGCOLOR="#XXYYZZ">

XX	Es un número indicativo de la cantidad de color rojo
YY	Es un número indicativo de la cantidad de color verde
ZZ	Es un número indicativo de la cantidad de color azul

Estos números están en numeración hexadecimal. Esta numeración se caracteriza por tener 16 dígitos (en lugar de los diez de la numeración decimal habitual). Estos dígito son:

0 1 2 3 4 5 6 7 8 9 A B C D E F

Es decir, que en nuestro caso, el número menor es el 00 y el mayor el FF. Así, por ejemplo, el color rojo es el #FF0000, porque tiene el máximo de rojo y cero de los otro dos colores. Los colores primarios son:

#FF0000	Rojo
#00FF00	Verde
#0000FF	Azul

Otros colores son:

#FFFFFF	Blanco
#000000	Negro
#FFFF00	Amarillo

Para hacer un color más oscuro, hay que reducir el número de su componente, dejando los otros dos invariables. Así, el rojo #FF0000 se puede hacer más oscuro con #AA0000, o aún más oscuro con #550000.

Para hacer que un color tenga un tono más suave (más pastel), se deben variar los otros dos colores haciéndolos más claros (número más alto), en una cantidad igual. Así, se puede convertir el rojo en rosa con #FF7070.

Colores del texto y de los enlaces

Si no se variasen los colores habituales del texto y de los enlaces (negro y azul, respectivamente), podría ocurrir que su lectura contra un fondo oscuro fuese muy dificultosa, o incluso imposible, si el fondo fuese precisamente negro o azul.

Para evitar esto, se pueden escoger los colores del texto y de los enlaces, añadiendo a la etiqueta los siguientes comandos:

TEXT	color del texto
LINK	color de los enlaces
VLINK	color de los enlaces visitados
ALINK	color de los enlaces activos (el que adquieren en el momento de ser pulsados)

Los códigos de los colores son los mismos que los que se han visto anteriormente.

La etiqueta, con todas sus posibilidades, sería:

```
<BODY BGCOLOR="#XXYYZZ" TEXT="#XXYYZZ" LINK="#XXYYZZ" VLINK="#XXYYZZ"
ALINK="#XXYYZZ">
```

El comando TEXT explicado anteriormente (que va englobado dentro de la etiqueta <BODY>) cambia el color de la totalidad del texto de la página.

Tanto el Netscape Navigator 2, como el Microsoft Explorer soportan una etiqueta de color de la fuente con la que se puede cambiar sólo una parte del texto:

<FONT COLOR="#XXYYZZ"> Este texto es de color XXYYZZ </FONT>

1.8 Tablas

Las tablas pueden parecer un modo sencillo de disponer el texto en columnas o quizás de añadir un titular a una ilustración, pero hay modos de sacar un gran partido de una característica aparentemente sencilla. La etiqueta <TABLE> puede ser una poderosa herramienta de formato. Se puede hacer por ejemplo, no mostrar el borde de una tabla en absoluto. También se puede hacer uso de la etiqueta <TABLE> para ubicar texto e imágenes con precisión, en prácticamente casi cualquier lugar de una página.

1.8.1 Estructura de una tabla

Vamos a ver ordenadamente (de fuera hacia dentro) las etiquetas necesarias para confeccionar las tablas.

<TABLE> [resto de las etiquetas] </TABLE>	Es la etiqueta general, que engloba a todas las demás.
<TABLE BORDER=n> [resto de las etiquetas] </TABLE>	Presenta los datos tabulados con un borde, haciendo las tablas más atractivas, y el grosor es de n píxeles.
<TR> [etiquetas de las distintas celdas de la primera fila] </TR>	Permite formar cada fila de la tabla. Hay que repetirla tantas veces como filas se desea que tenga la tabla.
<TD> [contenido de cada celda (imágenes, texto, etc.)] </TD>	Permite formar las distintas celdas que contendrá cada fila de la tabla. Hay que repetirlas tantas veces como celdas se desea que tenga la fila.
<TH> [encabezamiento de tabla] </TH>	Es utilizada para colocar encabezamientos en negrita sobre las columnas

(Ver anexos-ejemplo 4)

1.9 Formularios

La manera general para que los lectores de una página dada se puedan comunicar con la misma es por medio de un enlace a la dirección de e-mail del propietario de la página, con lo que recibiría un e-mail convencional.

Pero puede ser que lo que se necesite sea solamente una respuesta concreta a unas opciones que sean presentadas por el constructor de la página, o un comentario del usuario, para lo sería necesario suministrar un espacio en donde introducirlo.

Se puede, hacer todo esto, además de otras cosas, utilizando los formularios, con los que se pueden confeccionar páginas que contengan los elementos necesarios para ello, tal como botones de radio, listas de selección, cajetines de introducción de texto y de control, etc.

Los formularios permiten que los demás envíen la información directamente al servidor fuente, en donde se ha instalado un programa que procese esta información. Por ejemplo, si se necesita crear una lista de correo. Los usuarios pueden introducir sus nombres y direcciones de e-mail y pulsar un botón de envío.

Esos datos se los puede recibir "en bruto" en el correo destino, con los que se contruiría manualmente dicha lista de correo, sin necesitar ningún programa para ello.

1.9.1 Estructura de un formulario

La estructura general de un formulario es:

1. Etiqueta de inicio:
   <FORM ACTION="mailto:dirección_de_email" METHOD="POST"
   ENCTYPE="TEXT/PLAIN">
2. Cuerpo del formulario
   (Elementos para introducir los datos).
3. Botones de envío y de borrado.
4. Etiqueta de cierre </FORM>

1.9.2 Etiqueta de inicio

El atributo ACTION indica la acción que se debe efectuar y que es que los datos sean enviados por e-mail a la dirección indicada. (Si se hiciera uso del CGI, sería precisamente aquí donde se indicaría su localización en el servidor, que habitualmente es el directorio cgi-bin, para que procese los datos).

El atributo METHOD=POST indica que los datos sean inmediatamente enviados por correo a la dirección de e-mail, nada más púlsar el usuario el botón de envío.

Con el atributo ENCTYPE="TEXT/PLAIN" se consigue que las respuestas se las reciba como un archivo de texto, perfectamente legible y sin codificar.

1.9.3 Cuerpo del Formulario (Elementos para introducir los datos)

La introducción de los datos se consigue por medio de la etiqueta:

<INPUT TYPE= “XXX” NAME=“YYY” VALUE= “ZZZ”>

En donde:

XXX Es la palabra que indica el tipo de datos a introducir.
YYY Es el nombre que se le asigna a la variable de introducción del dato.
ZZZ Es la palabra asociada a un elemento.

1.9.3.1 Introducción por medio de texto

Introducción por medio de texto (una línea)

En este caso es XXX=TEXT, es decir, INPUT TYPE=“TEXT”. El atributo VALUE no procede en este caso. A continuación un ejemplo.

Se solicita el apellido del usuario:
<FORM ACTION=mailto:direccion_de_e-mail METHOD=“POST” ENCTYPE=“TEXT/PLAIN”>

Escribe tu apellido:

<INPUT TYPE=“TEXT” NAME="Apellido">
</FORM>
Si el usuario introduce su apellido, p. ej. Arias, y pulsa el botón de envío, se recibirá, un e-mail suyo con el siguiente texto:
Apellido=Arias

La longitud de este formulario es por defecto de 20 caracteres. Se puede variar incluyendo en la etiqueta el atributo SIZE="número". Por otra parte, sea cual sea la longitud del formulario, si no se indica nada, el usuario puede introducir el número de caracteres que quiera. Se puede limitar esto, incluyendo en la etiqueta el atributo MAXLENGTH="número".

En el caso que ya se ha visto, si se hubiera cambiado la etiqueta correspondiente por:
<INPUT TYPE="text" NAME="Apellido" SIZE="10" MAXLENGTH="12">

(Se puede comprobar cómo no se pueden introducir más de 12 caracteres).

También se puede hacer que el texto introducido no sea reconocible, es decir que todos los caracteres se representen por asteriscos. Basta con cambiar en la etiqueta INPUT TYPE=“TEXT” por INPUT TYPE=“PASSWORD”. En el último ejemplo, si se cambia la etiqueta correspondiente por:

<INPUT TYPE="PASSWORD" NAME="Apellido" SIZE="10" MAXLENGTH="12">

1.9.3.2 Introducción por medio de texto (múltiples líneas)

Cuando el texto a introducir puede alcanzar una gran longitud, por ejemplo un comentario, es conveniente utilizar un formulario de texto de múltiples líneas.

Esto se consigue con la etiqueta de inicio:

<TEXTAREA NAME=“YYY” ROWS="número" COLS="número">

(en donde no se utiliza INPUT TYPE y donde ROWS representa el número de filas, y COLS el de columnas).

y la de cierre: </TEXTAREA>
Ejemplo: un formulario solicitando los comentarios del usuario:

```
<FORM ACTION=“mailto:direccion_de_e-mail” METHOD="POST"
ENCTYPE="TEXT/PLAIN">

Introduce tus comentarios:
<BR><TEXTAREA NAME="Comentarios" ROWS="6" COLS="40">
</TEXTAREA>
</FORM>
```

(El salto de línea del texto introducido no se efectúa automáticamente).

Una vez que el usuario haya escrito sus comentarios dentro del formulario, y haya pulsado el botón de envío, se recibirá un e-mail suyo con el siguiente texto:
Comentarios = mensaje escrito por el usuario

1.9.3.3 Introducción por medio de menús

Si se desea que el usuario, en vez de introducir un texto, como ya se ha visto en los casos anteriores, escoja entre varias opciones que se le presenta, puede hacer uso de un formulario en forma de menú.

Se consigue con la etiqueta de inicio <SELECT NAME= “YYY”> y la de cierre </SELECT>.

Las distintas opciones a escoger se consiguen con la etiqueta <OPTION>.
Ejemplo: Se pide al usuario que elija su color preferido:

```
<FORM ACTION=“mailto:dirección_de_e-mail” METHOD= “POST”
ENCTYPE=“TEXT/PLAIN”>

<BR>¿Cuál es tu color preferido?
<BR><SELECT NAME="ColorPreferido">
<OPTION SELECTED>
<OPTION>Rojo
<OPTION>Verde
<OPTION>Azul
<OPTION>Amarillo
</SELECT >
</FORM>
```

Si el usuario ha escogido, p. ej. Azul y ha pulsado el botón de envío, se recibirá un e-mail suyo con el texto: ColorPreferido=Azul.

En el ejemplo anterior, sólo es visible en el formulario una opción. Si se desea que sean visibles múltiples opciones a la vez, se puede añadir en la etiqueta los atributos MULTIPLE SIZE="número", donde se especifica el número de opciones visibles.

Si se cambia en el ejemplo anterior la etiqueta correspondiente por:

<SELECT NAME="ColorPreferido" MULTIPLE SIZE="2">

1.9.3.4 Introducción por medio de botones

Caja de confirmación (checkbox)

Si se quiere que el usuario confirme una opción determinada, se puede hacer uso de un formulario de confirmación, o checkbox, que se consigue con la etiqueta:

<INPUT TYPE=“CHECKBOX” NAME=“YYY”>

Ejemplo: Se solicita al usuario que confirme su inclusión en una lista de correo:

<FORM ACTION= “mailto:dirección_de_email” METHOD=“POST” ENCTYPE=“TEXT/PLAIN”>

<INPUT TYPE="checkbox" NAME="Lista">
Sí, deseo ser incluido en la lista de correo.
</FORM>

Si el usuario marca este formulario y pulsa el botón de envío, lo que se recibe es un e-mail suyo con el texto: Lista=On.

Si se quiere que el formulario aparezca inicialmente como marcado (el usuario no necesitará hacerlo), basta con añadir el atributo CHECKED dentro de la etiqueta. En el ejemplo anterior se substituye la etiqueta equivalente por:

<INPUT TYPE=“CHECKBOX” NAME=“Lista” CHECKED>

1.9.3.5 Botones de radio

Cuando se necesita que el usuario elija una única opción entre varias, se puede hacer uso de los botones de radio, que se consiguen con la etiqueta:

<INPUT TYPE= “RADIO” NAME= “YYY” VALUE= “ZZZ”>

Donde YYY es el nombre que le se le coloca a la variable que se trata de elegir, y ZZZ es el nombre de cada una de las opciones en concreto.

Ejemplo: Se solicita al usuario que defina cuál es su sistema operativo preferido:

```
<FORM ACTION=“mailto:dirección_de_email” METHOD=“POST” ENCTYPE=“TEXT/PLAIN”>

¿Cuál es su sistema operativo preferido?
<BR>
<INPUT TYPE="radio" NAME="SistemaOperativo" VALUE="PC" CHECKED> PC
<INPUT TYPE="radio" NAME="SistemaOperativo" VALUE="Mac"> Mac
<INPUT TYPE="radio" NAME="SistemaOperativo" VALUE="Unix"> Unix
</FORM>
```

Se puede observar que el atributo opcional CHECKED que se ha añadido en la primera etiqueta. Esa será la opción que aparece marcada por defecto.

Se puede observar también que no es posible escoger más de una opción.

Si el usuario ha escogido la opción PC y pulsa el botón de envío, se recibirá un e-mail suyo con el texto: SistemaOperativo=PC.

1.9.3.6 Botones de envío y de borrado

Hasta ahora, en todos los ejemplos que se ha visto, faltaba un elemento esencial en cualquier formulario, y es el botón de envío de los datos, que se consigue con la etiqueta:

<INPUT TYPE= “submit” VALUE= “ZZZ”>

En donde ZZZ es el texto que se necesita en el botón.

Vamos a añadirlo al primer ejemplo, en el que se solicitaba el apellido del usuario:

```
<FORM ACTION="mailto:dirección_de_email" METHOD="POST" ENCTYPE="TEXT/PLAIN">

Escribe tu apellido:
<BR><INPUT TYPE="text" NAME="Apellido">
<P><INPUT TYPE="submit" VALUE="Enviar datos">
</FORM>
```

Otro botón interesante es el de borrado de los datos introducidos, muy conveniente en un formulario con muchos elementos. Es muy similar al de envío, pues se consigue con la etiqueta:

```
<INPUT TYPE= “RESET” VALUE=“ZZZ”>
```

En donde ZZZ es el texto que se necesita en el botón.

Si se añade al ejemplo anterior la etiqueta:

```
<P><INPUT TYPE=“reset” VALUE=“Borrar datos”>
```

Se puede comprobar su funcionamiento, escribiendo algo en el formulario y pulsando luego el botón de borrado.

Consideraciones finales

Hasta ahora se ha visto uno a uno los diferentes elementos que se pueden utilizar. Pero no hay ningún inconveniente en usar, dentro del mismo formulario, distintos tipos de introducción de datos. Al pulsar el usuario el botón de envío se recibiría en e-mail suyo con las distintas parejas NAME=VALUE de cada elemento, encadenadas con el símbolo &. (Ver Anexos- ejemplo 5)

2. Clientes Dinámicos

2.1 Programación Orientada a eventos

Cuando se crea un programa tradicional, que se ejecuta desde la línea de comandos, el programa asume que es lo único que se está ejecutando y que no tiene porqué interactuar con un entorno más grande. De modo que empieza a ejecutarse y termina, pudiendo ver el programador el hilo de ejecución que va a seguir sin ningún problema. En el caso particular de Java, la aplicación comienza cuando empieza el código del método main y termina cuando se llega al final de dicho método.

En cambio, si se sitúa un programa dentro de un entorno mayor, como puede ser una página web o un entorno multitarea como Windows o X-Windows lo que sucede es que el que se encarga de ejecutar las aplicaciones es dicho entorno. Y dado que son entornos donde pasan más cosas que la ejecución de nuestro programa, es necesario que éste no se ejecute de principio a fin, ocupando toda la capacidad de proceso de ordenador.

La aplicación no se ejecuta de principio a fin, sino que lo hace en respuesta a mensajes que le manda el entorno en el que se ejecuta. Estos mensajes se envían cuando suceden eventos que el entorno piensa que afectan a la aplicación. Por ejemplo, cuando el ratón pasa por encima de la misma, cuando hay que redibujarla porque se ha cerrado una ventana que tenía encima, etc.
Los applets funcionan de esa manera. Disponen de muchos métodos que se pueden sobrescribir y que se ejecutan al crearse el applet, al dejar de estar visible, cuando vuelve a estarlo, cuando el ratón pasa por encima de él, cuando hay que redibujarlo, etc.

2.2 Viewer de los Applets

Se puede utilizar un navegador web como Explorer, Netscape o HotJava para ver nuestros applets, especialmente cuando se quiere incluir dentro de una página web ya construida. Sin embargo, mientras se programe el applet, será recomendable utilizar el programa que viene incluido en el JDK para probarlos. El appletviewer recibe como parámetro una página HTML y nos muestra el applet incluido en la misma en acción.

Para utilizarlo, por lo tanto, se debe conocer como incluir applets en una página HTML.

2.3 Construcción de Applets

Un applet es una pequeña aplicación accesible en un servidor Internet, que se transporta por la red, se instala automáticamente y se ejecuta en un sitio como parte de un documento web. Dado que Java es multiplataforma y los applets son soportados por ambos navegadores, su uso se ha popularizado bastante.

Los applets tienen, sin embargo, un problema que impide que se utilicen más habitualmente. Este problema consiste en que, al igual que sucede con las imágenes, cuando se coloca un applet en una página web se debe definir un rectángulo de un tamaño determinado donde permanecerá el mismo, no pudiendo actuar fuera de sus márgenes.

2.3.1 Como crear un applet

A continuación el detalle.

HolaMundo.java

```
/**
 * Applet HolaMundo
 *
 * <APPLET CODE="HolaMundo.class" WIDTH="200" HEIGHT="70"></APPLET>
 */

import java.applet.Applet;
import java.awt.*;

public class HolaMundo extends Applet {
  public void paint(Graphics g) {
    g.drawString("  Bienvenidos a este mundo"  ,20,20);
  }
}
```

Lo podemos grabar cono HolaMundo.java y se ejecutael appletviewer con el siguiente comando:

```
appletviewer HolaMundo.java
```

```
import java.applet.Applet;
```

```
import java.awt.*;
```

En todos applet se debe, como mínimo, incluir estas dos sentencias de importación. En la primera se importa la clase Applet, de la cual se derivan nuestros applets. La segunda permite importar las clases de la librería AWT, que nos permitirán dibujar en el rectángulo asignado al applet.

```
public class HolaMundo extends Applet {
```

```
}
```

Un applet siempre van a ser clases derivadas de Applet, pues en esta clase y en sus clases padre se definen los métodos que se deben sobrescribir para que los applets respondan a los eventos generados desde el navegador.

```
 public void paint(Graphics g) {
```

```
 }
```

Este es el método que se debe sobrescribir para dibujar en el rectángulo asignado a nuestro applet. Recibe un parámetro que es una referencia a una instancia de tipo Graphics. Esta clase permite definir lo que se suele denominar lienzo (o Canvas).

```
  g.drawString("Hola, mundo ",20,20);
```

La clase Graphics dispone de muchos métodos para dibujar en el lienzo. Éste, concretamente, permite dibujar un texto en la posición que se quiere. En este caso el texto se pintará 20 pixels a la derecha del borde izquierdo de la pantalla y otros 20 más abajo del borde de arriba.

2.4 Ciclo de vida de un applet

Como se ha visto, un applet se ejecuta en una página web y ese entorno en el que está va a determinar el ciclo de vida del mismo. La clase Applet dispone de varios métodos sin código dentro a los que llama en diversas fases de su ciclo de vida (al crearse el applet, al redibujarse, al destruirse). Si se desea que un applet que se construya, haga cosas especiales en esos momentos se debe sobrescribir estos métodos.

void init()

Este método se llama al crearse el applet y sólo se ejecuta esa vez. Se suele utilizar para inicializar variables globales.

void start()

Se llama cuando hay que activar el applet. Esto sucede cuando el applet se ve en la pantalla. Se suele utilizar para comenzar las actividades de applet, ya que en general éste debe estar visible en la pantalla para poder realizar su función.

void stop()

Es el inverso del anterior. Es llamado cuando el applet deja de estar visible. Se suele utilizar para interrumpir las actividades activadas en start().

void destroy()

Método llamado al destruirse el applet. Normalmente no se suele sobrescribir, ya que la destrucción del applet conlleva la destrucción de todo lo que se ha creado en él. Puede ser útil para mostrar una ventana de confirmación que pregunte al usuario si de verdad debe cargárselo.

void repaint()

Este método no debería ser sobrecargado y es llamado cuando se necesita redibujar el applet. Puede ser llamarlo desde el código. Llama a update(Graphics).

void update(Graphics)

Llamado por repaint() cuando se necesita actualizar el applet. Su implementación borra el rectángulo y llama a paint() para volver a pintarlo.

void paint(Graphics)

Método en el que se dibuja el applet y el cual se debe sobrescribir para poder dibujar lo que se desea.

void print(Graphics)

Equivalente a paint() pero es llamado cuando se va a imprimir el contenido del applet.

El siguiente ejemplo conviene observarlo desde un navegador e ir viendo los distintos mensajes que nos muestra el applet, para comprender cuando son llamados los métodos.

2.4.1 Etiqueta Applet

La etiqueta APPLET presenta varios parámetros, de los cuales sólo es obligatorio poner los ya comentados CODE, WIDTH y HEIGHT. Son los siguientes:

CODE

Nombre completo (incluyendo extensión) del fichero que contiene el applet.

WIDTH

Anchura del rectángulo donde se ejecutará el applet.

HEIGHT

Altura del rectángulo donde se ejecutará el applet.

CODEBASE

Dirección donde está el fichero.class que contiene al applet. Es necesario ponerlo cuando el applet se encuentra en un directorio distinto al de la página desde la que se le llama, ya que CODE no puede contener directorios, sólo el nombre del fichero.

ALT

Algunos navegadores comprenden la etiqueta APPLET pero no pueden mostrar applets. Esto es debido a no tener instalada la máquina virtual Java o a que son

navegadores en modo texto. En ese caso mostrarán el contenido de este parámetro en lugar del applet.

2.5 Paso de parámetros

Entre <APPLET> y </APPLET> se puede colocar etiquetas PARAM que se permita pasar parámetros al applet. Tienen dos atributos:

VALUE

Nombre del parámetro.

NAME

Valor del parámetro.

Se Puede obtener esos valores por medio del método getParameter(String), como se ve en el ejemplo.

Hay que destacar que Java no distingue entre mayúsculas y minúsculas en cuanto al nombre de los parámetros.

En muchos casos, el usuario puede que no incluya parámetros que se consideran necesarios o que escriba mal el nombre de un parámetro. En ese caso, la llamada a getParameter() nos devolverá null. Se debe tener cuidado con esto, ya que nos pueden saltar excepciones por esta clase de cosas. Así pues, el código correcto de init() será:

```
public void init() {
  mensaje = getParameter("Mensaje");
  if (mensaje==null)
    mensaje = "Mensaje por defecto";
}
```

De este modo, si se comete una equivocación en la etiqueta PARAM, nos mostrará un mensaje y no lanzará ninguna excepción.

2.6 Clase Graphics

La clase Graphics dispone de más métodos aparte de drawString, que nos permitirán dibujar figuras e imágenes.

2.6.1 Métodos de dibujo

Mientras no se especifique, se considera que todos los parámetros son enteros:

drawString(String texto,x,y)

Escribe un texto a partir de las coordenadas (x,y).

drawLine(x1,y1,x2,y2)

Dibuja una línea entre las coordenadas (x1,y1) y (x2,y2).

drawRect(x,y,ancho,alto)
fillRect(x,y,ancho,alto)
clearRect(x,y,ancho.alto)

Son tres métodos encargados de dibujar, rellenar y limpiar, respectivamente, un rectángulo cuya esquina superior izquierda está en las coordenadas (x,y) y tienen el ancho y alto especificados.

drawRoundRect(x,y,ancho,alto,anchoArco,altoArco)
fillRoundRect(x,y,ancho,alto,anchoArco,altoArco)

Equivalentes a los anteriores, pero con las esquinas redondeadas. La forma y tamaño de dichas esquinas viene dada por los dos últimos parámetros.

draw3DRect(x,y,ancho,alto,boolean elevado)
fill3DRect(x,y,ancho,alto,boolean elevado)

Equivalentes a los primeros, pero dibujan un borde para dar la sensación de que está elevado o hundido (dependiendo del valor del último parámetro).

drawOval(x,y,ancho,alto)
fillOval(x,y,ancho,alto)

Dibujan una elipse con esquina izquierda superior en (x,y) y el ancho y alto especificados. Si son iguales dibujará un círculo.

drawArc(x,y,ancho,alto,anguloInicio,anguloArco)
fillArc(x,y,ancho,alto,anguloInicio,anguloArco)

Dibuja un arco cuyo primer vértice está en (x,y) y el segundo en (x+ancho,y+alto). La forma del mismo vendrá dado por los dos últimos parámetros.

drawPolygon(int[]coordenadasX,int[]CoordenadasY,numCoordenadas)
fillPolygon(int[]coordenadasX,int[] coordenadasY,numCoordenadas)

Dibuja un polígono cerrado del número de puntos especificado en el último parámetro.

copyArea(xOrigen,yOrigen,ancho,alto,xDest,yDest)

Copia el área cuya esquina superior izquierda está en (xOrigen,yOrigen) y de ancho y alto especificados a la zona que comienza en (xDest, yDest).
Por ejemplo, se puede dibujar lo siguiente.

2.6.2 Clase Color

Los dibujos necesitan contener color para realzar su objetivo de presentación, Graphics almacena internamente el color con el que pinta todo, que por defecto es negro. Para poder cambiar dicho color se debe utilizar el método setColor(), que recibe un único parámetro de tipo Color.

La clase Color dispone de varias propiedades estáticas que contienen los colores más comunes; son Color.black (negro), Color.blue (añil), Color.cyan (azul), Color.darkGray (gris claro), Color.gray (gris), Color.green (verde), Color.lightGray (verde claro), Color.magenta (magenta), Color.orange (naranja), Color.pink (rosa), Color.red (rojo), Color.white (blanco) y Color.yellow (amarillo). Pero también se pueden crear instancias de color por medio de constructores para definir colores menos comunes. Por ejemplo:

Color c = new Color(int rojo, int verde, int azul)

El color creado tendrá las cantidades de rojo, verde y azul indicadas, las cuales deberán estar en un rango que va de 0 a 255.

Color c = new Color(int rojo, int verde, int azul, int alfa)

Idéntico al anterior, pero añade un canal alfa que indica el grado de transparencia. Sólo funciona con el JDK 1.2.

Se puede observar lo que se obtiene cambiando el ejemplo anterior

2.6.3 Manejo de imágenes

Para utilizar imágenes se debe utilizar el siguiente método:

drawImage(Image img,int x,int y,ImageObserver observador)

ImageObserver es un interfaz que nos ofrece información sobre la imagen según la imagen se carga de la red. Por otro lado, Image es un objeto que representa a la imagen. Normalmente se crea un objeto Image por medio del método de Applet getImage():

Image img = getImage(URL base, String fichero)

Asigna la imagen contenida en la URL donde se encuentre el fichero que contiene la imagen que se quiere presentar y el nombre de ese fichero.

Normalmente se obtendrá esa dirección a partir de la página donde esté la página HTML o nuestro applet. Para averiguarlo se dispone de dos métodos en la clase Applet:

URL getDocumentBase()

Devuelve la URL donde está la página HTML (excluyendo el nombre del fichero).

URL getCodeBase()

Devuelve la URL donde está el applet (excluyendo el nombre del fichero).
Si se desea obtener una URL distinta se debe importar esa clase (java.net.URL) y crear uno pasándole una cadena como parámetro al constructor conteniendo la URL.

Normalmente, se realiza el getImage() en el método init() del applet, ya que en el momento en que se ejecuta este método comenzará la carga de imágenes desde

Internet y eso conviene hacerlo cuanto antes. Para mostrar la imagen se utiliza el método paint(), al igual que se ha hecho para pintar cosas.

2.7 Ejecución multihilo en applets

Hasta ahora sólo se disponía de un hilo de ejecución. Es decir, en los programas ejemplo sólo se ejecuta una cosa al tiempo. Si se desea que se ejecuten varias cosas a la vez o simplemente se desea disponer de una manera precisa de controlar la ejecución de parte del programa dependiendo del tiempo, se necesita que el applet realice una ejecución multihilo.

El concepto de multitarea es diferente al de multihilo. El primero se suele aplicar a la ejecución concurrente de programas distintos, ya que para que exista multitarea se exige que cada proceso ejecutándose disponga de su propio espacio en la memoria. En el segundo caso, en cambio, todos los hilos que se ejecutan concurrentemente disponen del mismo espacio de memoria.

El ejemplo siguiente es una modificación del applet MostrarMensaje, realizado anteriormente, que realiza un scroll lateral del texto, que aparece por la derecha y desaparece por la izquierda. El código nuevo que concierne a la ejecución multihilo está en negrita para facilitar su identificación:

Lo primero es asegurarnos que nuestro applet implementa el interfaz Runnable, necesario para el soporte de threads, o hilos de ejecución. Después se declara un objeto de tipo Thread, es decir, hilo, y se lo iguala a null. En el método start(), que es donde debe estar todo el código de inicialización, se crea el hilo y se lo arranca llamando a su método start(). En stop(), por el contrario, se para la ejecución.

El método run() es nuevo y es consecuencia de la implementación del interfaz Runnable. Cuando se crea un hilo de ejecución, como se ha realizado en start(), el hilo recién creado siempre comienza su vida útil ejecutando el método run(). Así pues, aquí estará el código del hilo. Se comienza preguntando si existe nuestro hilo (hilo!=null) y si está ejecutándose (isAlive).
Luego mueve el texto un poco a la izquierda, repinta, y se obliga a sí mismo a dejar de ejecutarse durante 10 milisegundos por medio del método sleep(). Transcurrido ese tiempo seguirá su ejecución por el mismo sitio donde la dejó (de ahí que sea necesario el while).

Indicar por último que es obligatorio capturar la excepción InterruptedException para poder llamar al método sleep(). En caso contrario el compilador se queja. Así pues, se puede hacer, aunque no se especifique ningún código en caso de que se lance dicha excepción. Esto ocurrirá en casos muy excepcionales, por lo que no se debe preocupar.

2.8 Técnica Double-buffering

Cuando se llama al método repaint(), éste se encarga de llamar a update(), el cual borra lo que haya en el applet y llama a paint() para pintarlo de nuevo. Y todo eso lo contemplan nuestros ojos. El parpadeo es debido a que nuestro ojo se da cuenta de que se ha borrado la pantalla y que se tarda cierto tiempo en escribir de nuevo el texto.

Para remediarlo existe una técnica llamada double-buffering. Esta técnica no es exclusiva de Java, pues por ejemplo muchos de los juegos que utilizamos la implementan, o incluso utilicen técnicas más complicadas. Este método consiste en construir una pantalla virtual, que no se visualiza pero que existe en la memoria del ordenador, y pintar en ella lo que se necesite mostrar para volcarla a la pantalla real cuando se lo necesite.

Eso elimina nuestros dos problemas. Primero, no se borra nunca la pantalla, sino que se la sustituye, por lo que la retina no aprecia que se quede en blanco en ningún momento. Segundo, el volcado es un proceso muy rápido, por lo que no se pueden apreciar que la pantalla virtual se ha volcado sólo un poco pero no como sucedía cuando se dibuja directamente.

Para implementar esta técnica, se debe tener en cuenta que Graphics es una clase abstracta, por lo que no se puede utilizar para crear esa pantalla virtual. En cambio, se utiliza Image, que dispone de un método que nos será muy útil en este caso:

Graphics img.getGraphics()

Este método creará un lienzo a partir de una imagen, con las mismas dimensiones que ésta.

También se puede utilizar el método de Applet llamado createImage(anchura, altura) que crea una imagen en blanco del tamaño especificado. Así pues, se modifica el ejemplo, ennegreciendo de nuevo el código nuevo que interviene en la implementación en Java de la técnica double-buffering:

Se inicializa los dos nuevos objetos necesarios para implementar esta técnica en el método init() utilizando como tamaño de la imagen y su lienzo asociado el del applet. Este tamaño se lo averigua por medio de una llamada a getBounds().

El proceso de implementar la técnica se realiza en la sobre escritura del método update(). Este método recibe como parámetro el lienzo correspondiente al rectángulo que ocupa el applet en la pantalla. Sin embargo, lo que hace es limpiar el lienzo de la pantalla virtual y llama a paint utilizando como parámetro este último lienzo, por lo que paint() dibuja en la pantalla virtual. Por último, vuelca el buffer en la pantalla real.

Esto último es posible debido a que buffer y pantallaVirtual son referencias que apuntan a objetos que en realidad representan lo mismo. De modo que todo lo que se haga al lienzo pantallaVirtual se ve reflejado en la imagen buffer y viceversa.

2.9 Eventos

La interacción con el usuario se obtiene en Java por medio de eventos. Cuando sucede un evento que afecta al applet, la máquina virtual de Java ejecuta un método que se llama controlador de eventos. En principio esos métodos no tienen nada, por lo que se debe sobrescribirlos en función de nuestras necesidades.

Esos métodos son:

boolean mouseDown(Event, int, int)

Llamado cuando se pulsa un botón del ratón; los parámetros segundo y tercero contienen la posición del ratón cuando ocurrió el evento, mientras que el primero en un objeto que contiene información adicional sobre el mismo.

boolean mouseUp(Event, int, int)

Llamado cuando se suelta el botón del ratón después de haberlo pulsado.

boolean mouseMove (Event, int, int)

Llamado cuando se meve el ratón.

boolean mouseDrag (Event, int, int)

Llamado cuando se mueve el ratón con un botón pulsado.

boolean mouseEnter (Event, int, int)

Llamado cuando el ratón entra en el applet.

boolean mouseExit (Event, int, int)

Llamado cuando el ratón abandona el applet.

boolean keyDown(Event,int)

Llamado cuando se pulsa una tecla. El argumento entero indica cuál.

boolean keyUp(Event, int)

Llamado cuando se suelta una tecla.

Como ejemplo, se puede hacer un ojo que sigue al ratón.

El controlador de evento mouseMove se ejecuta cuando el ratón se mueve por encima del applet. Es en ese momento cuando se modifica la situación de la pupila del ojo, siguiendo al puntero del ratón.

2.10 Documentación

En muchas ocasiones, se distribuyen los applets haciendo públicos sólo los ficheros .class. Para informar de ciertos datos al usuario se dispone de dos métodos que deben ser sobrescritos:

String getAppletInfo()

Devuelve una cadena que identifique el applet.

String[][] getParameterInfo()

Devuelve un array bidimensional, donde la primera dimensión es el número de parámetros y la segunda es 3. Cada parámetro dispone, por tanto, de tres cadenas para detallar su información: la primera indica el nombre del mismo, la segunda su rango de valores y la tercera su utilidad. (ver ejemplo 6 y 7)

3. Clientes Multimedia

3.1 Que es JMF

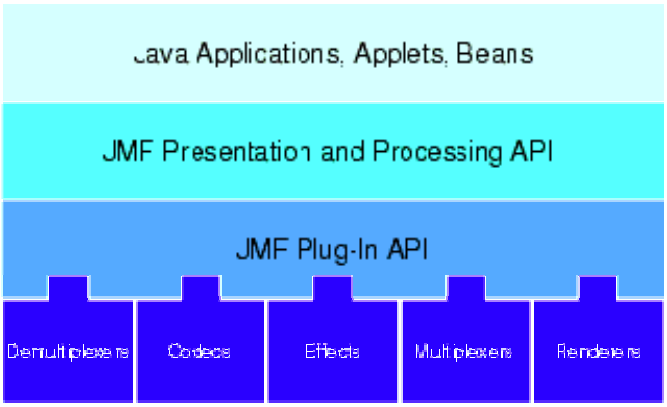
El API JMF es una definición del interfaz utilizado por Java para la utilización de multimedia y su presentación en *Applets* y aplicaciones. Como definición de interfaz, JMF no tiene por qué proporcionar las clases finales que manejan los datos multimedia, pero dice cómo los proveedores de dichas clases deben encapsularlas y registrarlas en el sistema.

Para el tratamiento de multimedia en casos especiales existe además otras APIs complementarias a JMF y más simples o mejor adaptadas a otros problemas.

Para el caso del sonido el Java Sound API permite la captura, el tratamiento, el almacenamiento y la presentación de sonido, tanto en formato MIDI como sonido muestreado, aunque no se pueden tratar todos los formatos existentes de almacenamiento y transmisión de sonido. En el caso particular de la voz, JSAPI (Java Speech API) define la interfaz existente para trabajar con sintetizadores y reconocedores de voz.

Históricamente, JMF nació con el objetivo de poder representar datos multimedia en los programas pero, en las últimas versiones las capacidades se extienden a todo tipo de tratamiento como la adquisición, procesado y almacenaje de datos multimedia, así como la transmisión y recepción a través de la red mediante el protocolo RTP.

JMF. ha sido desarrollada por Sun e IBM y no está incluida en las especificaciones de Java 2 o de la máquina virtual, por lo que es necesario obtener el paquete adicional que contiene el JMF para la plataforma que se esté utilizando.



3.1.1 Conocimientos Generales

Antes de empezar el estudio de cada uno de los componentes del JMF hay que aclarar algunos conceptos relativos a la transmisión multimedia.

En todo tratamiento que se pueda hacer con los datos multimedia siempre existen tres pasos, estos son: la adquisición de datos (captura desde un dispositivo físico, lectura de un fichero o recepción desde la red), procesado (aplicación de efectos como filtrado o realces, compresión y/o descompresión, conversión entre formatos) y la salida de datos (presentación, almacenamiento en fichero o transmisión a través de la red).

Los datos deberán estar en un formato definido como WAV, QuickTime, MPEG, etc, vengan de la fuente que fuere este tipo siempre debe estar definido.

El flujo de datos (o *media strea*m) puede contener varios canales (o *track*s) de datos, cada uno de los canales puede tener un tipo de datos distinto, así puede haber un *media stream* con video y audio contenido en dos *tracks* distintos. Un *media stream* queda perfectamente definido por su localización y el protocolo necesario (HTTP, FILE, etc.) para acceder a él. Los *media streams* se pueden dividir en tipos *push* y *pull* cuando el servidor o el cliente respectivamente controlan el flujo de datos. Como ejemplos de los primeros pueden estar RTP, VoD o la captura de sonido desde el micrófono, ejemplos del segundo caso son los protocolos HTTP y FILE o la captura de imágenes de una webcam.

Para prevenir interrupciones, parte de los datos se almacenan en un buffer antes de la presentación, por este motivo la presentación no es inmediata existiendo un cierto tiempo de retardo o latencia.

A continuación se presentan las clases e interfaces más comunes en JMF.

3.2 Visión General

3.2.1 Clase DataSource.

Esta clase se encarga de encapsular la localización, el protocolo de transferencia y el software necesarios para la adquisición. Por lo tanto una vez obtenido un DataSource no se puede asociar a otra fuente de datos.

Se construye a partir de un URL o un MediaLocator. Un MediaLocator es como un URL pero se puede construir aunque no se encuentre instalado en el sistema el *handler* para el protocolo especificado.

Para la construcción de estos objetos hace falta localizar los datos, por ello su creación por parte del programador sería muy complicada. Para facilitar este extremo, lo que se hace es pedir al sistema el DataSouce necesario a partir de la URL o el MediaLocator a través de la clase Manager como se verá más adelante en el punto.

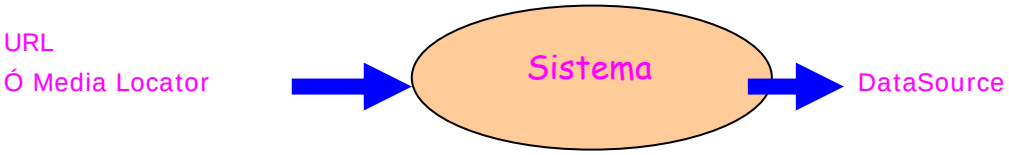


Diagrama 1. Creación del DataSource

3.2.2 Clases Player y Processor.

Un Player es un objeto encargado de procesar cierto flujo de datos multimedia y presentarlo en su preciso momento. La forma de presentación y los recursos necesarios dependen del tipo de media que contenga el flujo.

Como en el caso anterior, no se utiliza el constructor para crear objetos de esta clase sino que se llama a una función (createPlayer()) que busca en el sistema los componentes adecuados y crea el Player que se necesita en cada momento.

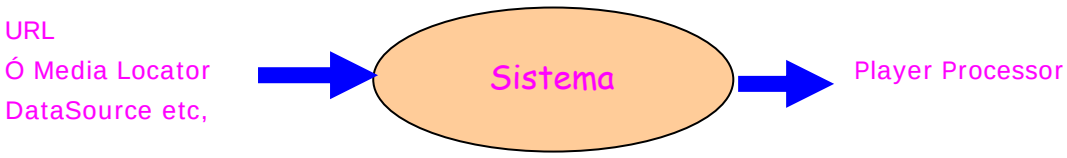


Diagrama 2. Creación del Player o Processor

Para obtener el flujo de datos se utiliza un objeto DataSource que se “engancha” a la entrada del Player. En el caso de que a la función createPlayer() se le pase un URL o un MediaLocator, esta construye primero el DataSource y luego el Player.

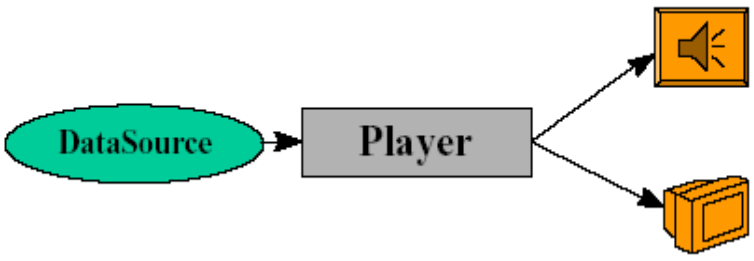


Diagrama 3. Flujo de Datos en Player

Para utilizar un Player es necesario que obtenga recursos del sistema por lo que pasa por varios estados antes de poder reproducir los datos multimedia, sin embargo, el usuario de JMF no debe preocuparse de esta gestión ya que hay funciones que en un

sólo paso crean un Player y le asignan recursos. Aquí se puede observar la gran potencia de este interfaz ya que si lo único que se quiere es añadir media a un programa basta con crear el objeto a partir de una fuente de datos y reproducir los datos multimedia, sin embargo usuarios más avanzados podrían controlar el proceso de la obtención de recursos, liberándolos cuando no sean necesarios, y, por último, programadores expertos podrán crear los Players a medida.

En cuanto a los objetos Processor, estos son hijos de Player, por lo que lo contado anteriormente también es aplicable a esta clase de objetos. Las únicas diferencias estriban en que la función que crea el Processor es distinta (createProcessor()) y que se le añaden nuevas capacidades.

Un Processor puede presentar los datos multimedia como hace el Player, pero es interesante saber que la salida puede ser un objeto DataSource que proporciona un *media stream* a otros objetos, gracias a esto se puede cambiar el formato de los datos multimedia o incluso procesarlos dentro del Processor.

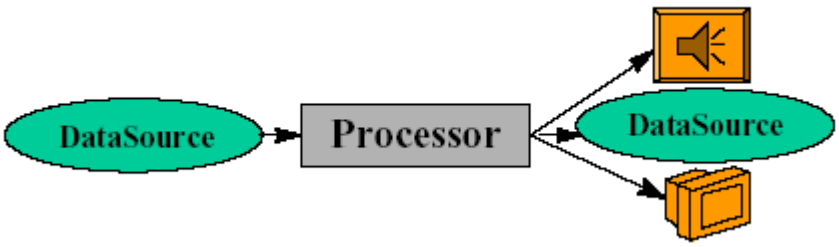


Diagrama 4. Flujo de Datos en Processor

3.2.3 Clase DataSink.

El último paso en el proceso de los datos multimedia puede ser el almacenamiento en algún fichero o la transmisión. Para estos casos JMF proporciona la clase DataSink. Un objeto de esta clase, como en los casos anteriores, se construye a través de la clase Manager usando un DataSource. La función de este objeto es obtener el *media stream* y almacenarlo en un fichero local, a través de la red o transmitirlo mediante RTP.

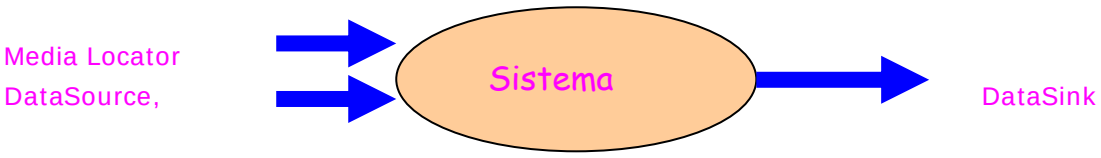


Diagrama 5. Creación de DataSink

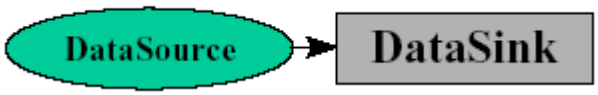


Diagrama 6. Flujo de Datos en DataSink

3.2.4 Manager y otras clases de control.

Debido a la complejidad del sistema se presentan algunos problemas como el hecho de que no se puedan utilizar constructores para todos los objetos, o el desconocimiento por parte del programador de las clases existentes en el sistema que heredan de las clases básicas JMF. Surge la idea de las clases *manager*s, que facilitan el uso de los demás tipos de objetos.

La más básica de estas clases es la clase Manager, esta es una clase intermedia para facilitar la construcción de los componentes JMF descritos anteriormente (Player,Processor, DataSource y DataSink). El tener esta in dirección permite crear,desde el punto de vista del usuario, de igual manera componentes ya existentes o nuevas implementaciones.

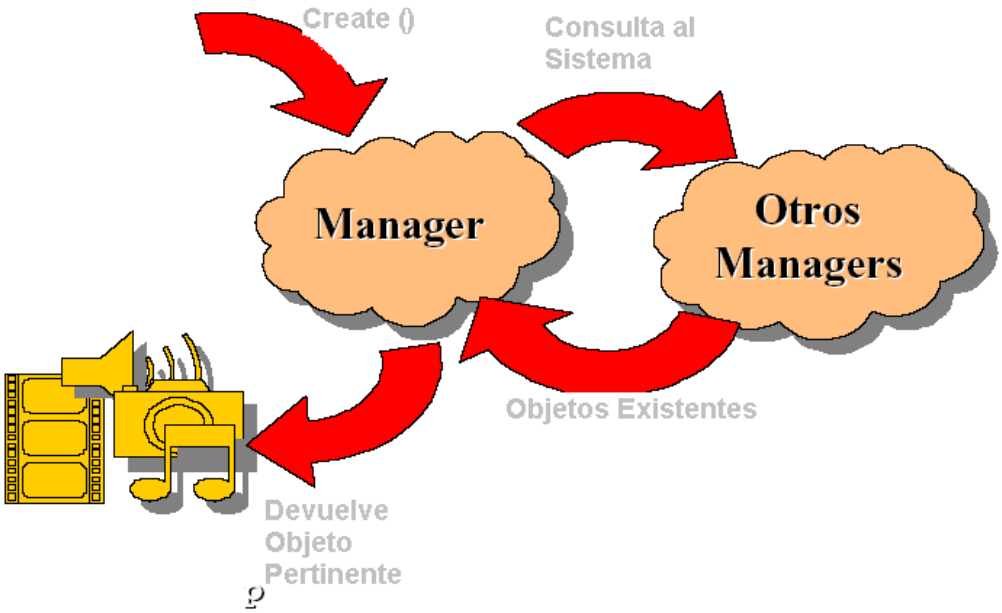


Diagrama 7. Funcionamiento de Manager

Para mantener un registro de los paquetes que pueden tener clases JMF como Players, Processors, DataSources y DataSinks se utiliza el PackageManager. CaptureDeviceManager mantiene registro de los dispositivos de captura de datos multimedia del sistema. Por último existe el PlugInManager que lleva la cuenta de los *plug-ins* existentes en el sistema. Un *plug-in* es un componente capaz de procesar los datos, como anticipo se puede decir que son los que componente un Player o un Processor, pueden ser Multiplexer, Demultiplexer, Codec, Efecto Renderer

3.2.5 Otras clases importantes.

Estas clases no son las que constituyen la parte más importante del sistema pero son necesarias para definir o controlarlas estas clases son, por ejemplo Format, MediaEvent, EventListener, Controls, ProcessorModel, MediaLocator, MediaError,MediaException, Control, TimeBase, etc.

Format sirve para definir el tipo de media, existen las clases hijas AudioFormat y VideoFormat (que a su vez se divide en más) para adaptarse mejor.

Estas clases tienen constantes (atributos final static) que definen los formatos más comunes. Son necesarias para definir el formato de salida de Processor, obtener el formato de un *media strea*m, construir un ProcessorModel, etc.

MediaEvent es la clase padre de todos los eventos lanzados por los componentes JMF, como en otros casos sirven para advertirnos de pequeños errores, anunciar el fin de una actividad (muy útil para comunicar el fin de la presentación de media), comunicar el paso de un estado a otro en un objeto (necesario en Player y Processor), etc. Estos eventos siguen los patrones *Java Beans* establecidos permitiendo realizar fácilmente una comunicación con otros objetos del sistema (por ejemplo para cerrar la ventana de visualización de un vídeo al terminar). Para los diferentes tipos de eventos existen diferentes EventListener.

ProcessorModel es una clase capaz de definir internamente un Processor, cada uno de los *tracks* y tipos de datos internos, se usa para la creación de un Processor con unas características muy definidas o para analizar un Processor existente.

El objeto MediaLocator define la localización de la fuente de datos y el protocolo a utilizar para obtenerlos, a diferencia de un URL, no necesita que el sistema implemente dicho protocolo para la creación del objeto. MediaError y MediaException son los padres de todos los errores y las excepciones que los objetos JMF pueden lanza cuando ocurre un error.

Control es una clase diseñada para controlar las características del flujo de datos de un *track* al procesarse, dependiendo del tipo de datos el objeto Control puede ofrecer unas u otras características, por ello esta clase se divide en herencia en otras muchas. Hay que tener cuidado y no confundir la clase Control con la clase Controls que es la definición de los dispositivos que componen un Player. TimeBase representa la base de tiempos que tienen todos los objetos Clock (Player y Processor) la modificación de la velocidad o el momento de comienzo y final se obtiene respecto a esta base temporal. Para realizar una sincronización entre varios Player estos deben compartir sus elementos TimeBase.

3.3. Visión en Detalle

3.3.1 Jerarquía DataSource.

La jerarquía de esta clase se puede ver a continuación. Por otra parte, a diferencia de algunos lenguajes como UML, la flecha que indica herencia apunta hacia la

clase hija y está rellena. Además no hay distinciones entre clases e interfaces Java ya que para comprender el sistema es indiferente.

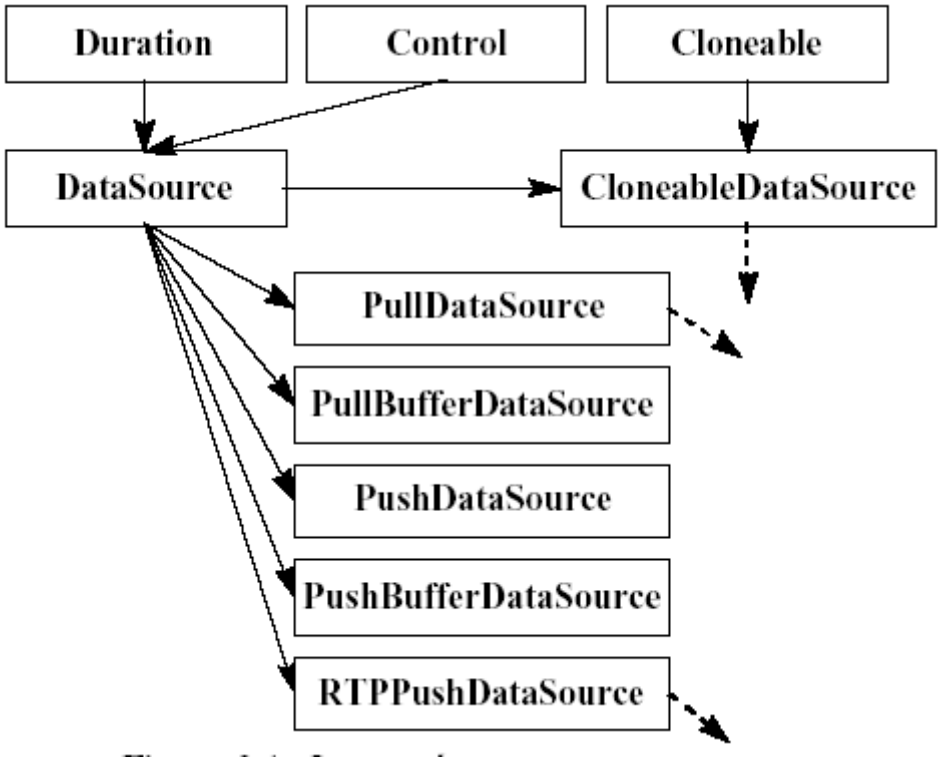


Diagrama 8. Jerarquía DataSource.

3.3.1.1 Padres.

El primero de los padres de DataSource que se observa es Controls, esta interfaz tiene un mecanismo para poder ajustar las características del objeto, DataSource usa este interfaz para proporcionar acceso a los objetos Control que tenga como atributos.

El otro padre es Duration. Un objeto Duration tiene una duración definida, a la que se puede acceder con la función getDuration(), con esto se indica la duración total del *media strea*m, en algunos casos esta duración puede ser DURATION_UNKNOWN (cuando es desconocida) o, en el caso de una transmisión en directo, DURATION_UNBOUNED.

3.3.1.2 Hijos.

En la parte de los descendientes de DataSource se puede ver dos partes, por un lado están los que descienden solamente de DataSource y por otro CloneableDataSource y sus hijos.

En el primero de los casos se puede dividirlos en los de tipo *Push* y los de tipo *Pull* y entre los que tienen *Buffer* y los que no. La diferencia entre *Push* y *Pull* radica en el tipo de media stream La diferencia entre tener *Buffer* y no tenerlo tiene que ver con el tipo de unidad de datos que se utiliza en al flujo de datos, esta puede ser una unidad simple como es el caso de una imagen o un *buffer* con varias muestras de la señal simple. Hay que señalar la existencia de RTPPushDataSource que se utiliza para la transmisión de datos a través de la red utilizando RTP, de esta clase hereda RPTSocket que facilita enormemente el uso de este protocolo para la transmisión multimedia en la red.

Respecto a las clases que derivan de Cloneable hay que decir que permiten realizar copias del objeto, esto en caso de DataSource es de gran utilidad al permitir duplicar el *media stream* para diferentes propósitos. Existen clases derivadas de CloneableDataSource equivalentes a las comentadas en el punto anterior.

3.1.1.3 Implementar nuevos DataSource.

Cuando se quieren utilizar nuevos tipos de datos multimedia es necesario implementar nuevas clases SourceStream y DataSource, esta nueva clase debe heredar de alguna de las ya existentes (PushDataSource, PullDataSource, PushBufferDataSource o PullBufferDataSource), si además implementa los interfaces Seekable o Positionable ofrecerá la posibilidad de saltar a un punto del *media stream* o cambiar de posición en él respectivamente.

Para que el sistema pueda registrarlo y luego pueda ser utilizado por otros objetos, hay cumplir dos condiciones. La primera de ellas es el nombre completo (incluido el *package*) del objeto este debe ser <prefijo del paquete>.media.protocol.<protocolo>.DataSource, así para el protocolo *ftp* se puede

crear la clase uazuay.edu.ec.media.protocol.ftp.DataSource. El otro requisito es registrar el prefijo del paquete en el sistema, para ello basta con obtener un Vector con los prefijos de los paquetes utilizando PackageManager.getProtocolPrefixList() , añadirle un nuevo String con el nombre del prefijo y salvarlo de nuevo.

3.3.2 Atributos de DataSource.

En este caso los atributos de DataSource están representados por la figura a continuación Cada línea de las que parten de DataSource significa "tiene", el número indica la cantidad de objetos que puede tener de cada tipo.

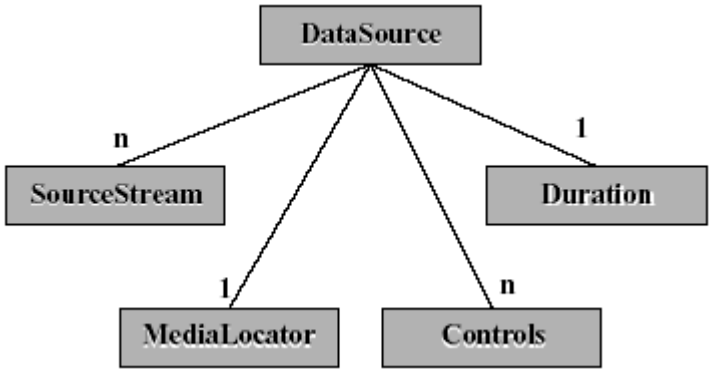


Diagrama 9. Atributos DataSource.

Como se puede ver un DataSource puede tener varios SourceStream, cada uno de ellos es una representación de los propios flujos de datos, dependiendo del tipo de DataSource los SourceStream pueden ser PullBufferStream, PullSourceStream, PushBufferStream o PushsourceStream.

MediaLocator ya fue descrito con anterioridad y, como se dijo, indica donde se encuentra la fuente de datos y que protocolo hay que usar para poder obtenerla.

Los objetos Control permiten modificar alguna de las particularidades de la señal, se dividen por herencia en un montón de tipos (20 clases) se puede obtener una lista de los existentes llamando a la función getControls() que implementa todo objeto Controls (como los DataSource).

Por último de Duration ya se ha hablado con anterioridad, encapsula la información sobre la duración del flujo de datos que se esté tratando.

3.3.3 Jerarquía de Player.

Es la que trata y/o representa los datos multimedia, por este motivo se presenta su diagrama de herencia antes de tratar los atributos o la parte dinámica del objeto comentando sus posibles estados.

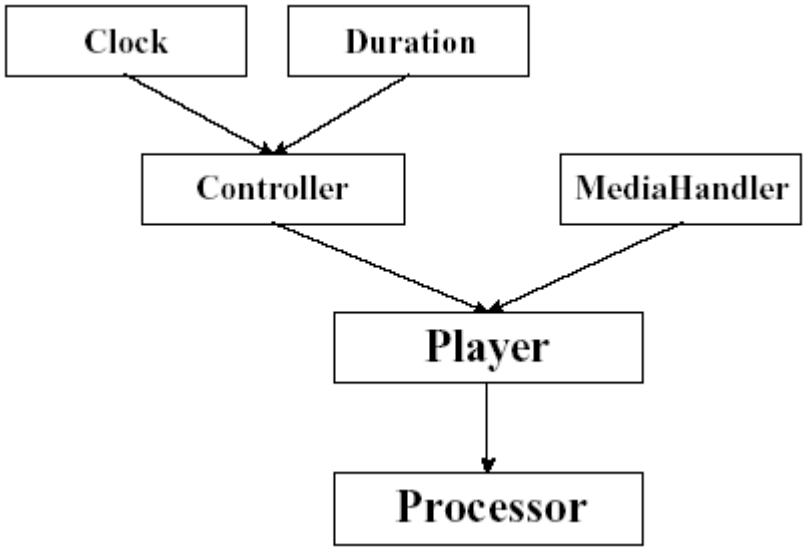


Diagrama 10. Jerarquía Player y Processor.

La primera clase que se ve en el diagrama de herencia es Clock que define las operaciones básicas de temporización y sincronización. Como ya se contó anteriormente esta clase tiene un objeto TimeBase, para poder manejarlo se utilizan las funciones getTimeBase() y setTimeBase() permitiendo la sincronización de varios elementos al obligarles a compartir la base de tiempos a todos ellos. Para controlar la velocidad de reproducción de los datos se usan las funciones getRate() y setRate(). Finalmente con getMediaTime() y setMediaTime(), que utilizan objetos Time, se puede manejar el tiempo actual de la reproducción.

Como ya se ha comentado el objeto Duration encapsula la información temporal del *media strea*m. Hay que comentar que Controller modifica su funcionalidad y, en lo referente al tiempo, cuando sincroniza varios Player (o Processor) se devuelven los datos del flujo más largo.

A continuación Controller aúna la funcionalidad de los anteriores, además define el modelo de eventos, estos eventos notificarán cambios en el flujo de datos, transiciones de estado de los objetos o eventos de terminación al acabar la reproducción o producirse un error.

Por otro lado MediaHandler define algunas funcionalidades de los elementos que derivan de él como la interfaz con objetos DataSource o elementos de creación y destrucción.

3.3.4 Atributos Player.

Los elementos de Player, como se muestra en la figura a continuación pueden ser muchos y muy variados debido a complejidad del sistema, a continuación se muestran los más importantes.

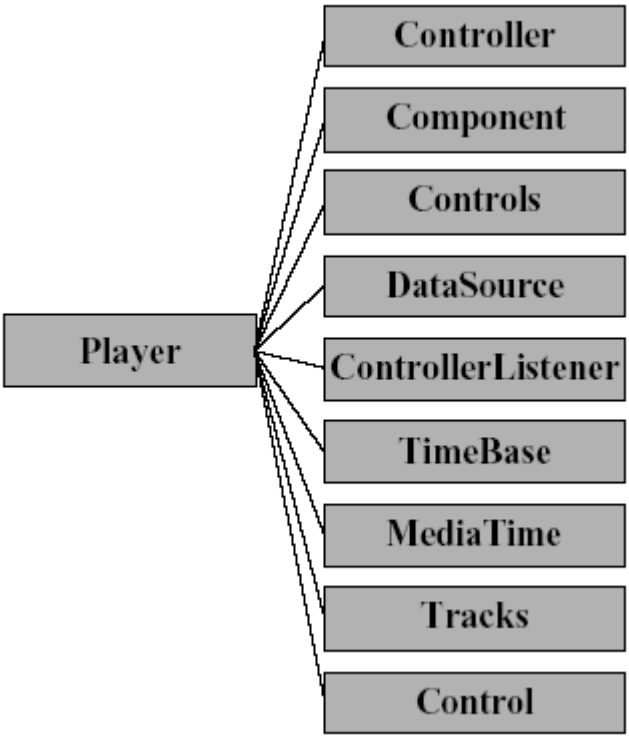


Diagrama 11. Atributos de Player.

3.3.4.1 Otros Controller: Sincronización de elementos.

El hecho de poder relacionar un Player con otro Controller permite que el primero pueda comunicarse con el segundo controlando parte de la funcionalidad, además comparten la base de tiempos permitiendo realizar la sincronización de forma sencilla.

En general para realizar la sincronización hay que compartir la base de tiempos, por ejemplo con una sentencia como p1.setTimeBase(p2.getTimeBase()), Además deben tener un ControllerListener que pueda controlar los diversos lanzamientos de eventos y hay que asegurarse de que el Rate (velocidad de presentación) es el mismo en todos los Player. Para empezar la reproducción hay que llevar a todos los Controller al mismo estado del modelo dinámico (por ejemplo a stop) y sincronizar las operaciones utilizando las funciones específicas para ello (por ejemplo hay que usar syncstart() en lugar de start()). Para poder actuar adecuadamente hay que llevar la cuenta del estado en el que se encuentra cada Controller.

Usando un Player para sincronizar otros Controller se simplifica mucho el asunto. Basta con añadir el Controller al Player con los métodos addController() y removeController() si se desea separarlos. A partir de aquí las acciones realizadas en el Player pasan a todos sus Controller, el Player mandará un evento cuando todos los Controller que tiene lo han mandado. El tener otros Controller hace que alguna característica cambie ligeramente, así la duración o la latencia será la máxima entre la del Player y las de los otros Controller.

3.3.4.2 Control temporal: TimeBase & MediaTime.

Por el hecho de ser de tipo Clock todo Controller, como los Player y los Processor, tienen un objeto TimeBase, que se encarga de proporcionar pulsos de reloj a velocidad constante como un cristal de cuarzo en un sistema hardware, y un objeto MediaTime que lleva la cuenta del tiempo de presentación del flujo de datos, de esta forma:

3.3.4.3 Tiempo de presentación: Duration.

Como ya se ha visto se puede obtener de Player un objeto Duration que muestra la duración del flujo de datos multimedia que se procesan.

3.3.4.4 Tratamiento de la señal: Controls y PlugIn.

Desde el punto de vista del tratamiento de datos un Player está compuesto por flujos de datos que van pasando a través de varios objetos transformándose hasta su consumo en la representación. Desde este punto de vista estos objetos, que son PlugIn yControls, son los más importantes del Player. En la figura a continuación se representa la jerarquía de Controls. Aquí se tratan los más importantes.

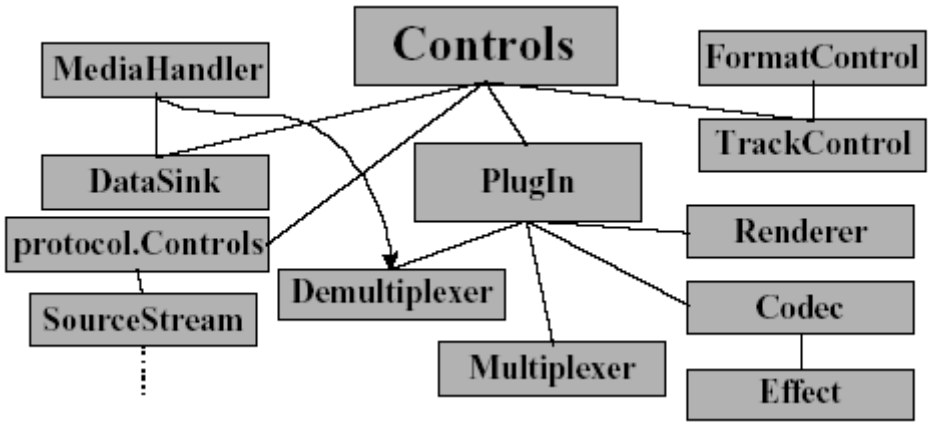


Diagrama 12. Jerarquía Controls.

En el estudio del Player los Controls más importantes son los PluIn, estos toman los datos los tratan adecuadamente y los dejan para el siguiente proceso. Entre ellos se encuentran los Demultiplexer que toman los datos de un *media stream* y los dividen en diferentes *track*s, sus principales métodos son setSource() donde se le especifica elDataSource de donde obtener los datos, getTracks() que devuelve los objetos Track representantes de los *tracks* extraídos del DataSource y getSupportedInputContendDescriptor() que devuelve información sobre los formatos de entrada soportados.

Los Codec son PlugIn que tratan la señal, codifican y/o decodifican cambiando, si procede, su formato. Los Effect son un caso específico de los Codec que realizan algún efecto o proceso a la señal. Los métodos de los Codec son getSupportedInputFormats() y getSupportedOutputFormats() que devuelven los formatos soportados como entrada y salida del Codec respectivamente, setInputFormat() y setOutputFormat() sirven para definir el formato de entrada y salida del Codec, por último process() realiza el proceso en el *track* correspondiente.

En el camino de la señal a través de un Player el último paso es un Renderer, cada uno de estos objetos es un PlugIn que representa el dispositivo físico que muestra el multimedia. Sus métodos son getSupportedInputFormats() que devuelve una lista con los formatos de entrada que soporta el objeto, setInputFormat() define el formato a presentar y process() que presenta los datos del *track* en el que se ha incluido.

Con todos estos objetos el Player ya es capaz de presentar los datos multimedia. La disposición de estos objetos es la mostrada a continuación

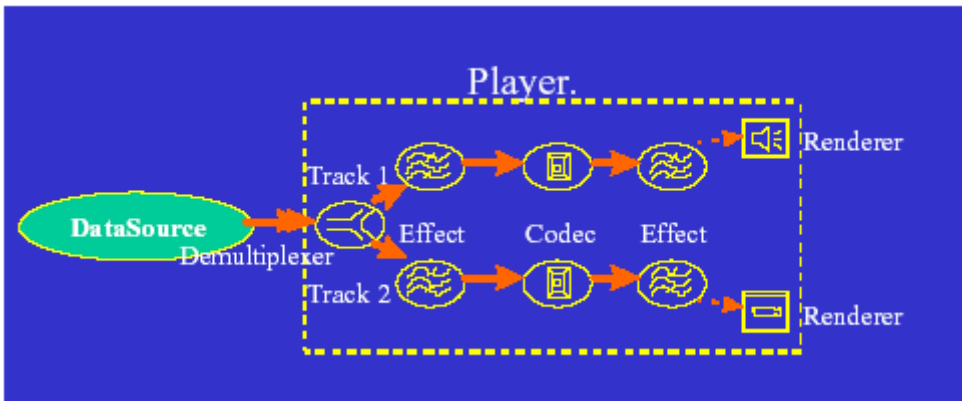


Diagrama 13. Disposición de Controls en Player.

Además de estos PlugIns, en un Processor puede haber un cuarto tipo, el Multiplexer que, al contrario que el Demultiplexer, se encarga, con los datos de varios *track*s, de obtener el flujo de datos correspondiente que formará el *stream* del DataSource a la salida del Processor. Los métodos de Multiplexer son getSupportedOutputContentDescriptor() que advierte de los formatos de salida soportados, setOutputContentDescriptor() que selecciona el formato de salida, process() convierte los *tracks* individuales en el *stream* de salida. Hay que decir que un DataSink también puede realizar la multiplexación por lo que no siempre este elemento es necesario.

Al añadir esta posibilidad la distribución de los objetos por los que pasan los datos queda como en la Figura mostrada a continuación

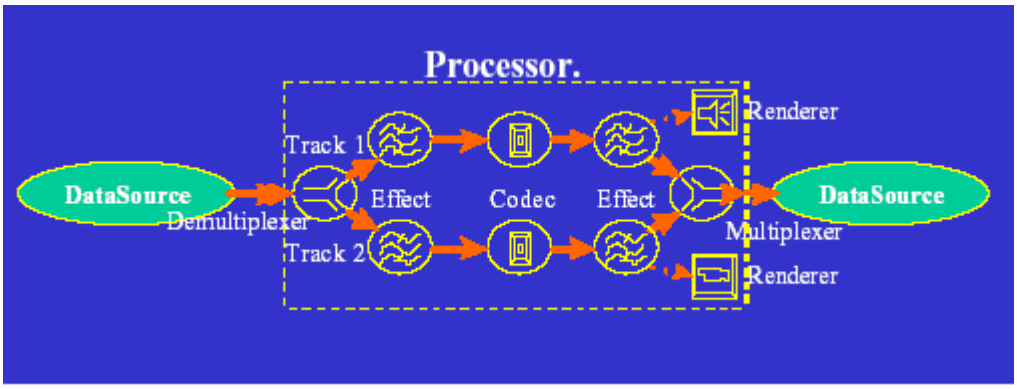


Diagrama 14. Disposición de Controls en un Processor.

Un Processor, además de los Controls ya comentados, puede tener uno o varios TrackControl, con este objeto se puede configurar cada uno de los caminos internos de la señal. Sus métodos setFormat(), que especifica las conversiones en el *track*, setCodeChain(), que selecciona el PlugIn usado, y setRenderer(), que selecciona el "visor", hacen que el usuario pueda configurar totalmente un Process.
Para obtener el TrackControl de cada Track sólo hay que invocar al método getTrackControls() de Processor.

Aunque quedan fuera del Player y el Processor, también hay que señalar que dentro de los Controls se encuentran los objetos de clase DataSink y SourceStream.

3.3.4.5 Características del proceso: Control.

Lo primero que se debe decir es que no hay que confundir la Clase Controls, que procesa el flujo de datos, con Control que una clase de objetos que cambia o presenta alguna característica del *strea*m. Se obtiene del Player con getControls() que devuelve una lista con los existentes. Estos Control pueden formar parte de DataSources o PlugIns. hay varios tipos de Control, los más comunes son CachingControl que informa de la cantidad de datos leídos de la fuente (muy útil cuando se obtienen los datos de la red), GainControl que puede alterar el volumen del audio, BitRateControl permite modificar la velocidad de transmisión, BufferControl muestra el estado del *buffer* y permite realizar operaciones simples en él, etc.

3.3.4.6 Control visual del proceso: Component.

Estos objetos son *AWT* Component, lo que significa que se pueden incluir en la ventana de una aplicación o en un Applet. Al ser componentes definidos en *AWT* se puede cambiar las propiedades y manipularlos con objetos LayOut. Puede haber otros Component, incluso los objetos Control pueden tener este tipo de objetos, pero los más corrientes son ControlPanelComponent y VisualComponent. Para obtener estos objetos se recurre a los métodos del Player (o el Processor) getControlPanelComponent() y getVisualComponent().

ControlPanelComponet mostrará al usuario los botones y controles que le sirvan para el manejo de la señal como pueden ser botones de reproducción y pausa, barra de volumen, etc.

VisualComponent muestra al usuario la representación de la propia señal o sus características, así en una señal de vídeo este objeto es la ventana donde se visualiza el vídeo, en una señal de audio puede verse la forma de onda, un panel de vúmetros o puede no existir.

3.3.4.7 Control del objeto: ControllerListener.

Como el modelo de eventos sigue los patrones habituales, el ControllerListener funciona como en las demás APIs, par ayudar a la programación existe la clase ControllerAdapter con la declaración trivial de todas las funciones.

Esta interfaz es necesaria para poder llevar el control del estado de un Player en el modelo dinámico, esta importancia es incrementada por el hecho de que las funciones que llevan de un estado a otro no son síncronas y los métodos regresan sin haber terminado su función. En el caso de sincronizar “manualmente” varios Controller es necesario el control de los eventos que puedan lanzar para tomar las medidas oportunas en el resto de Controller. También es importante cuando se almacena un flujo de datos la escucha de los eventos con el fin de cerrar el fichero al terminar. En otra ocasión en que se hace importante esta escucha es cuando se desea liberar recursos si no se están utilizando.

3.3.4.8 Origen y destino de la señal: DataSource.

Todo MediaHandler debe tener una entrada con el flujo de datos como ya se vio antes, en un Player esto se hace mediante un DataSource. En el Processor, además,

puede haber otro DataSource a la salida, para obtener este último se utiliza el método getDataOutput().

3.3.5 Modelo dinámico de Player y Processor.

Aunque es posible utilizar un Player y reproducir un flujo de datos sin conocer el modelo dinámico de este objeto, es conveniente estudiar dicho modelo para poder comprender el funcionamiento del objeto.

Por ser Clock un Player tiene dos estados Started y Stoped. Como esto no es suficiente la clase Controller ya divide Stop en cinco estados que, junto con Started son los que tiene Player.

Cuando se crea un Player con el método Manager.createPlayer() este se encuentra en su primer estado: Unrealized y lo único que tiene determinado es el DataSource. Con el método Realize() pasa al estado Realizing, en este nuevo estado el Player determina que recursos necesita (sólo los determina, no los captura) para la reproducción. Cuando se ha completado con éxito esta acción el Player manda un evento (RealizeCompleteEvent) pasando al siguiente estado: Realized.

En Realized ya se conocen los recursos necesarios, en este estado ya se pueden visualizar los controles, acceder a los datos del objeto y modificar sus características.

Para pasar al siguiente estado (Prefetching) se llama al método asíncrono (vuelve inmediatamente) prefetch(), en este estado se van adquiriendo los distintos recursos necesarios, cuando se termina se envía el evento PrefetchCompleteEvent pasando al estado Prefetched donde ya se está dispuesto a realizar la reproducción.

Tras llamar al método start() se empieza la reproducción entrando en el estado Started. Con el método Stop se deja de reproducir y se pasa del estado Started a Prefetched. Tanto si ocurre así como si se termina la reproducción por otro motivo al pasar de Started a Stop se lanza un evento (StopEvent). Si estando en los estados Prefetched o Prefetching se cambian los parámetros del Player o se invoca al método deallocate() se pasa al estado Realized. Liberando los recursos que se hayan obtenido (esto es muy útil cuando no se pretende utilizar el objeto en un tiempo elevado). En caso de utilizar deallocate() en Realizing se pasaría a Unrealized.

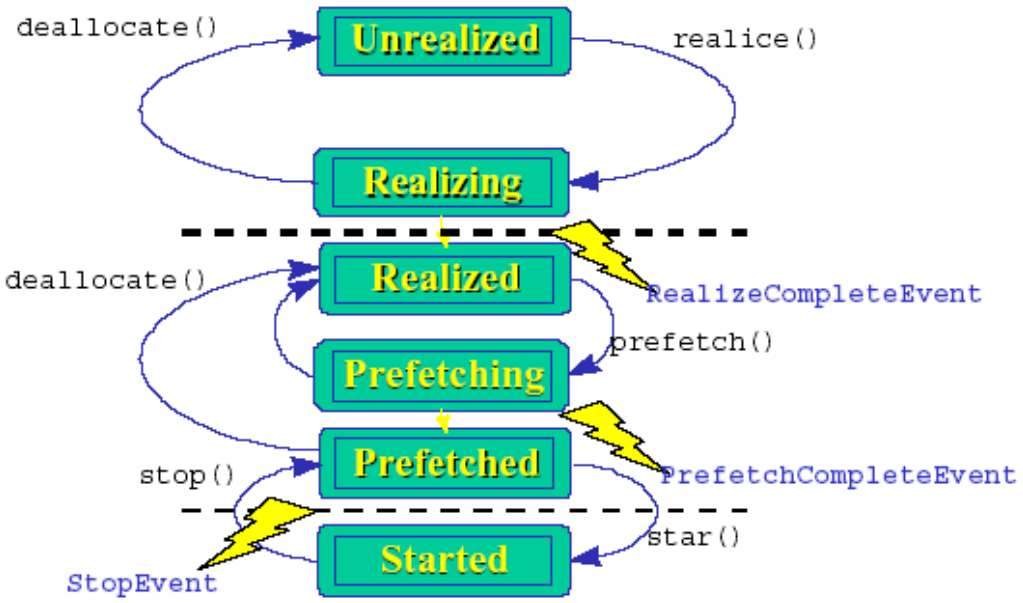


Diagrama 15. Estados de Player.

Es importante tener en cuenta que el utilizar funciones en los estados en los que no están permitidas hace que el método lance una excepción.

Por último hay que comentar que no hace falta pasar por todos los estados, así, en una primera aproximación para la reproducción de audio basta con crear un Player y llamar al método start(), este método hará que el Player pase por todos los estados hasta reproducir el sonido. En otros casos (por ejemplo vídeo) se puede crear el Player con el método de Manager createRealizedPlayer() que devuelve el Player en el estado Realized, se puede obtener aquí los controles y componentes visuales necesarios e invocar, como antes, al método start().

En el caso del Processor los estados Unrealized y Realizing se dividen en Unrealized, Configuring, Configured y Realizing. El primero de los estados es idéntico al del Player pero con la función configure() se pasa al estado Configuring donde se van conectando los componentes, como esta función es asíncrona, y retorna tras su llamada, para avisar del fin de esta operación el Processor lanza un evento (ConfigureCompleteEvent) antes de pasar al estado Configured, que es el siguiente.

En este estado se pueden obtener con getTrackControls() referencias a los distintos controles de cada *track* pudiendo configurar cada uno de ellos (por ejemplo

con setFormat()). Si se llama en Unrealized o en Configured al método realized() del Processor se pasa al estado Realizing que termina esta parte del proceso al igual que ocurría en el Player.

La secuencia de estados en el Processor queda como se muestra en la Figura A continuación

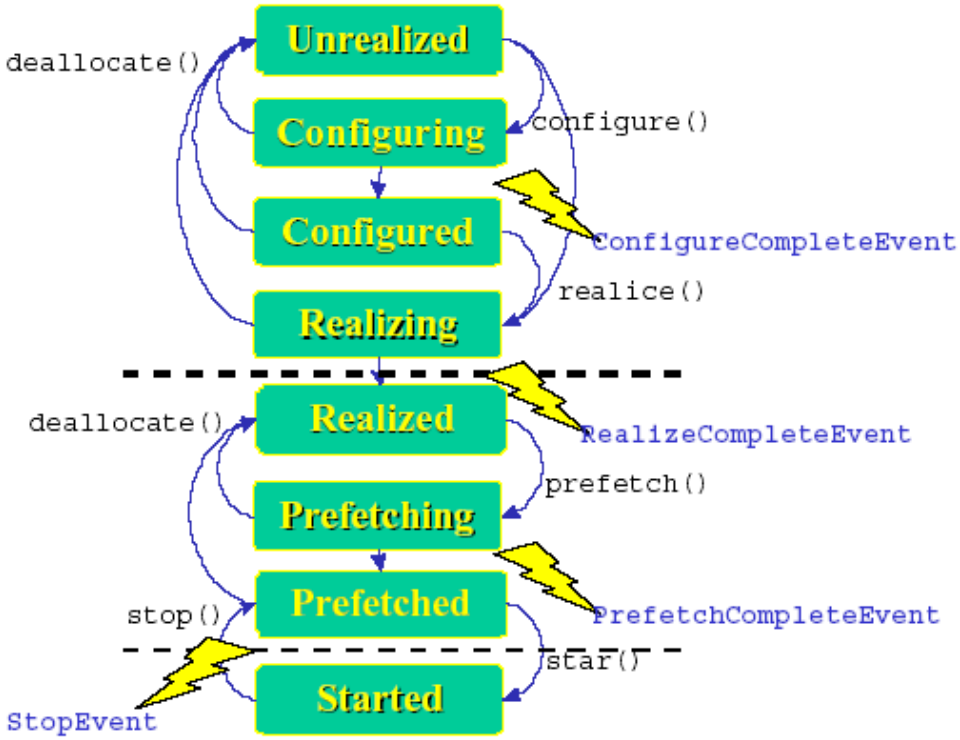


Diagrama 16. Estados del Processor.

Otra forma de configurar el Processor es utilizar un objeto ProcessorModel que define completamente la estructura del Processor (en cuanto al flujo de datos se refiere). Este objeto se pasa a la función de la creación del *manager* que construirá el Processor con estas características no siendo necesaria la configuración manual.

3.4 Utilización y control de los objetos.

Ya se ha comentado como se utiliza cada objeto, sin embargo hay opciones adicionales que pueden ser de gran interés, por otro lado en este capítulo se intentará ver desde un punto de vista más práctico algunas de las funciones comentadas.

3.4.1 Reproducción de multimedia.

En este caso lo primero es crear el Player, aunque se puede hacer con un DataSource lo más simple es utilizar el URL o MediaLocator directamente. Si no se utiliza createRealizedPlayer() se puede incluir un ControllerListener en el Player y llamar a realize(). Cuando el método controllerUpdate() de ControllerListener reciba un evento RealizeCompleteEvent se pueden obtener los componentes VisualComponet y ControlPanelComponent (con las funciones getVisualComponet y getControlPanelComponent()), asegurarse que los componentes existen y añadirlos en la ventana principal de la aplicación.

Si fuese necesario se puede comprobar si existen otros controles intentando obtenerlos con los métodos getGainControl(), getCachingControl(), etc, o bien consiguiendo una lista de controles con getControls(). A cada uno de esos controles se puede "pedirle" su componente visual con getControlComponent() y presentarlo si existiese.

Para fijar otras características de la reproducción se puede usar el método setRate() al que se le pasa un float con la velocidad relativa a la que se quiere la presentación (si el número es negativo irá hacia atrás en el tiempo, si es 0 permanecerá en pausa, si es positivo y menor que 1 irá a "cámara lenta", si es 1 la reproducción será normal y si es mayor irá más deprisa), hay que comentar que los Player no tienen por qué soportar cualquier número en este método; con setMediaTime() se puede modificar la posición inicial de la reproducción; etc.

3.4.2 Procesado de multimedia.

En el caso de la configuración del Processor en el estado Configured se pueden utilizar funciones como setOutputContentDescriptor() que define el formato de salida del Processor o, si se quiere modificar "a mano" el Processor pedir al PlugInManager una lista con los PlugIn disponibles mediante getPlugInList() y en cada TrackControl utilizar

setCodecChain() para especificar el *plug-in* del *track* o setFormat() para especificar el formato de salida del *trac*k.

3.4.3 Captura.

Aquí se necesita la ayuda del *manager* CaptureDeviceManager para localizar el dispositivo. Para ello, pasando un objeto Format a la función de este *manager* getDeviceList(), se obtiene un vector con la información de los dispositivos existentes (encapsulada en objetos DeviceInfo) que pueden capturar señal y mostrarla en el formato deseado, se selecciona de esta lista el objeto deseado o bien, si se conoce el dispositivo, se puede obtener su información con getDevice() del mismo *manage*r. De una forma u otra se puede llamar al método getLocator() del objeto seleccionado obteniendo un MediaLocator útil para la creación de un DataSource, un Player o un Processor. A partir de aquí será el MediaHandler el que maneje el flujo de datos capturados.

3.4.4 Almacenamiento.

Es necesario crear un URL o MediaLocator del destino, por supuesto también es necesario el DataSource que servirá de fuente. Con estos dos objetos se puede llamar a createDataSink() de Manager obteniendo nuestro objeto destino. Ahora basta con abrirlo mediante open() y, con start(), empezar la operación de almacenamiento. En el caso de que el DataSource sea la salida de un Processor es necesario, para no perder información, llamar antes a la función start() del DataSink que a la función start() del Processor (el cual debe estar en el estado Started para poder sacar datos hacia el DataSource de salida y, por lo tanto, hacia el DataSink). Cuando el flujo de datos termina el DataSink envía un evento EndOfStreamEvent que será necesario atender para destruir el DataSink utilizando su método close(). En el caso de una captura de datos multimedia, al llamar a los métodos stop() y close() del Processor también se lanza el evento. En el supuesto de saber cuando se quiere terminar el almacenamiento no es necesario la implementación del Listener.

3.5 Otras posibilidades.

A parte de lo comentado JMF tiene otras posibilidades, se resumen algunas de estas características de la API JMF.

3.5.1 Herencia.

El hecho de poder heredar de las clases definidas en JMF va a permitir que se pueda ampliar las posibilidades del sistema o personalizar los objetos.

En el caso de heredar de PlugIn se puede realizar un procesado propio de la señal o implementar nuevos protocolos y formatos para el manejo de nuevos tipos de ficheros, *stream*s, DataSource, etc. además nos permitirán añadir objetos Control aumentando su funcionalidad. Para heredar de PlugIn, dependiendo del tipo que sea (Demultiplexer, Codec o Effect, Multiplexer o Renderer) hay que implementar una serie de funciones y después registrarla con ayuda de la clase PlugInManager, para ello hay que usar el método addPlugIn(), en caso de querer eliminar del registro algún elemento basta con llamar a remove() de la misma clase, para que los cambios sean perennes en el sistema hay que llamar a commit() que graba el resultado.

Para implementar nuevos DataSource hay que heredar de alguno de los hijos de esta clase, además hay que implementar el correspondiente SourceStream.

Esto permitirá implementar nuevos protocolos de adquisición de datos y manejar nuevos tipos multimedia. El nombre de las clases tiene que ser estándar para poder registrarse en el sistema, con un prefijo de paquete, seguido de media.protocol, el nombre del protocolo que implementa y el nombre de la clase que será DataSource, en el caso de los SourceStream están situados en un escalón más arriba en la jerarquía de paquetes (<prefijo de protocolo>.media.protocol) y no tienen un nombre concreto.

Estas son las clases más útiles ya que permiten añadir funcionalidad a JMF, implementar Player, Component o Control permitirán variar y personalizar JMF.

3.5.2 Objeto MediaPlayerBean.

MediaPlayerBeam es un *Java Bean* que encapsula JMF. Este es fácil de construir e implementa funciones como setMediaLocator(), que especifica la fuente del *strea*m, y start() y stop() para la reproducción además de las correspondientes por ser *Java Bea*n. Esto permite incluir este tipo de objetos en otros módulos, incluso utilizarlos por programadores que no conozcan JMF.

Ejemplo. En la actualidad existe un sinnúmero de aplicaciones para Java Media FrameWork, para dar una ídea general de lo que esta herramienta es capaz de hacer, se ha incluído en los anexos un ejemplo de un reproductor de media

player en el cual se utiliza todo lo explicado anteriormente. (Ver Anexos-Ejemplo-15)

3.5.3 RTP.

Hay que conocer la posibilidad de transmisión y recepción de *streams* por la red utilizando el protocolo RTP *(Real-time Transfer Protocol)*. Al ser transmisión en tiempo real este protocolo se apoya en UDP y permite realizar "*unicast*" cuando se realiza transmisión punto a punto o "*multicast*" cuando se realiza una transmisión a una red de distribución. Entre los servicios ofrecidos por RTP se encuentra la identificación del tipo de datos, la ordenación y sincronización de los datos (incluso cuando se reciben de varias fuentes), proporcionan control y monitorización de la transmisión, etc. JMF permite transmisión, reproducción y almacenamiento de este tipo de datos. Para ello existen objetos SesionManager que controlan toda la sesión, existe un modelo de eventos adicional, distintos flujos de datos, nuevas clases derivadas de Format, reproductores especiales, DataSink para la transmisión, RTPSocket para la comunicación, etc.

Diagrama 17. Recepción RTP

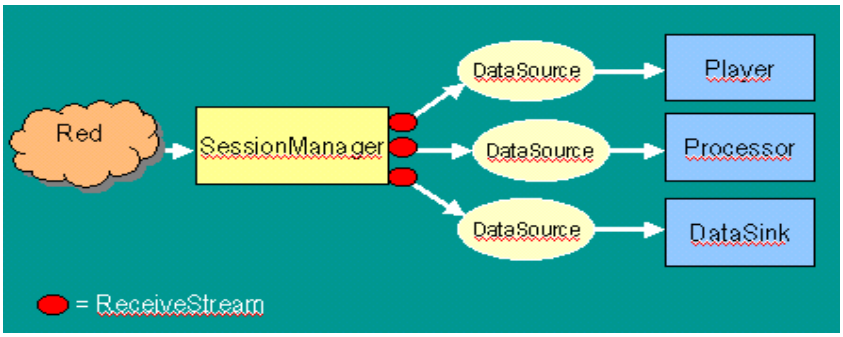
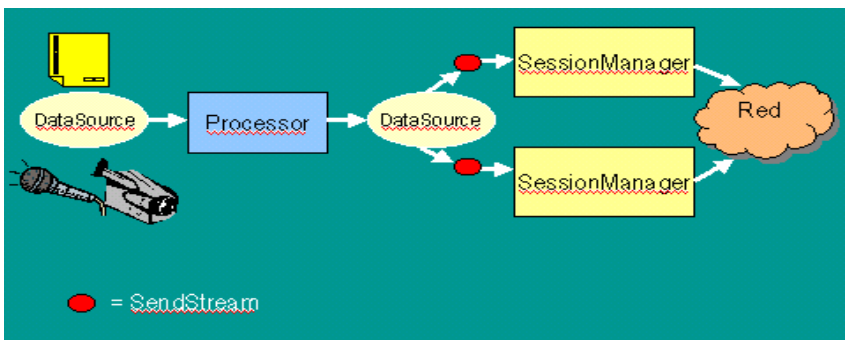


Diagrama 18. Transmisión RTP



4. Conclusiones

4.1 Resumen y conclusiones.

Como se conoce el lenguaje **HTML**, ***Hyper Text Markup Languaje*** o lenguaje de definición de marcas es un lenguaje sencillo que permite marcar los documentos de hipertexto mediante unas etiquetas especificas, de este modo conseguimos darle a los documentos una cierta estructura.

Para crear los documentos **HTML** podemos usar el block de notas de Windows por ser el más sencillo y común, pero se puede usar cualquier editor o procesador de texto para tal efecto, ya que este tipo de documentos tienen un formato de texto plano (***ASCII***).

Actualmente existen generadores de páginas web, que evitan el trabajo de escribir el código, sin embargo para un usuario de alto nivel es indispensable que tenga conocimientos básicos de HTML, además para enriquecer estas páginas se necesita con mayor razón conocer el funcionamiento de applets y por ende el conocimiento de Java, existen aplicaciones a todo nivel en las que no se necesita mayor conocimiento ya que existen un sin número de herramientas que generan el código de manera automática.

Como tenemos entendido un applet es una pequeña aplicación accesible en un servidor Internet, que se transporta por la red, se instala automáticamente y se ejecuta en un sitio como parte de un documento web Dado que Java es multiplataforma y los applets son soportados por varios navegadores, su uso se ha popularizado bastante.

Concluímos que El JMF consiste en una colección de tres APIs diseñados para captura y reproducción de audio y video, el cual nos brinda un sin número de ventajas para el manejo de multimedios.

Podemos anotar que es fácil de usar aplicaciones y applets pueden crear y controlar la reproducción de una gran cantidad de medios en diferentes formatos utilizando tal sólo pocos parámetros.

También soporte multiplataforma. Las aplicaciones y applets creados bajo el JMF pueden reproducirse sin ningún problema en cualquier plataforma que soporte java. JMF es distribuido en dos versiones, la primera es 100% Java y la segunda es una versión que utiliza clases nativas de la plataforma donde se encuentre instalada. El hecho de que cuenten con clases nativas hace que se optimice el manejo del audio y

video. Sin embargo en pruebas de desempeño la diferencia entre una versión y otra es muy pequeña.

Soporte para los formatos de audio y video más populares: JMF fue diseñado para suportar los formatos más comunes de audio y video tales como AIFF,AU, GSM, MIDI, MPEG, Quick Time, RMF y WAV.

A modo de resumen se puede decir que JMF es una API que permite, de forma muy simple, añadir multimedia a los programas Java. A su vez permite el tratamiento, la captura, el almacenamiento y la transmisión de este tipo de datos. El corazón de esta interfaz es el Player que permite la presentación de multimedia y, su hijo, el Processor que permite el tratamiento de la señal. Otros objetos importantes son los PlugIn que componen los Player y Processor encargándose de dividir, procesar, unir y presentar la señal.

Desde el punto de vista del sistema son muy importantes las clases *manager* que permiten la reutilización de los componentes desarrollados una vez por cualquier usuario, además tienen importancia el modelo de eventos con los MediaEvent y los correspondientes Listener para controlar el estado del sistema y el estado de cada objeto.

Otro punto importante de esta interfaz es que permite utilizarla desde varios puntos de vista, así un usuario básico puede no saber absolutamente nada del funcionamiento de multimedia y muy poco de JMF, sin embargo ser capaz de incluir elementos multimedia en sus programas utilizando los objetos básicos de este sistema. Un usuario experto tiene un punto de vista intermedio y puede llevar el control de los objetos JMF que cree así como de los estados del sistema, permitiendo realizar una programación óptima con el control de los recursos del sistema, presentación de los objetos más importantes en cada caso, etc. a su vez puede ser capaz de heredar de algunos objetos para personalizar la aplicación a su gusto.

4.2 Glosario.

API: *Application Program Interfac*e. Es la interfaz proporcionada por un sistema al programador para poder acceder a los servicios de ese sistema.

Applets:Aplicaciones de pequeño tamaño para ser incluidas en documentos.

Array: Es un grupo de datos de un tipo determinado puestos uno a continuación de otro. Coincide con los tipos de datos array de Java.

Browser: Un navegador es una aplicación cliente de software para Internet que sirve como interface para navegar a través del mundo de información del web. El primer navegador que existió se llamaba **Mosaic** y salió en 1993 y es el predecesor del Navigator**.**

IBM: Empresa norteamericana que ha desarrollado JMF junto a Sun.

FTP (File Transfer Protocol): El FTP sirve para "subir" o "bajar" (transferir) archivos entre computadoras y permite a los usuarios de Internet accesar computadoras remotas para obtener archivos de software o información. Es un protocolo de red.

Hipertexto: El hipertexto consiste en una combinación de texto común y corriente y algunas palabras, frases o íconos destacadas (generalmente subrayadas) de las demás; estas palabras se llaman ligas, enlaces o links y permiten ir fácilmente de un documento de hipertexto a otro; para seguir una liga basta con hacer "click" sobre ella con el mouse y de esta forma se "navega" a través de la información.

Host: Se refiere a una computadora conectada al Internet que cuenta con su propia dirección IP; el término nombre de host o hostname es el nombre de esta computadora. Un hostname completo describe el nombre de la computadora y el nombre de la red en la que se encuentra; con lo que se tiene la ruta completa.

Internet Message Access Protocol: Es un protocolo usado para accesar correo electrónico en máquinas remotas; un buzón tipo IMAP es una dirección de correo que almacena los mensajes y que permite consultarlos (leerlos, contestarlos, borrarlos, etc.) usando el algún browser (como pueden ser Internet Explorer o Netscape) y no un programa cliente email y los mensajes recibidos en este buzón siempre se guardan en el servidor donde reside el mismo.

Java: Lenguaje de programación orientado a objetos.

JDK: *Java Development Ki*t. Es el entorno de desarrollo para Java proporcionado por Sun.

JMF: *Java Media Famewor*k. API para el tratamiento de datos multimedia en Java.

JSAPI: *Java Speech AP*I. API para el tratamiento de voz y manejo de sintetizadores y reconocedores en Java.

MIDI: Estándar para el almacenamiento y transporte de música para o desde un sintetizador.

Package: Paquete. Agrupamiento especificado por el programador de clases con características comunes.

RTP: *Real-time Transfer Protoco*l. Protocolo de transferencia en tiempo real.

Real Audio: Es un protocolo de red que funciona bajo el esquema de cliente/servidor sirve para transmitir audio y/o video streaming en el web.

Sun: Empresa norteamericana que desarrolló el lenguaje de programación Java.

SGML - Standard Generalized Markup Language: Es un estándar internacional para la representación estructurada de información; usado para describir la estructura y contenido de documentos, basado en la idea de que los documentos cuentan con elementos estructurales y semánticos que pueden describirse sin referirse a cómo debe ser el estilo gráfico del documento; que podrá variar dependiendo del medio y las preferencias de estilo.

Thread: Hilo de ejecución.

URL (Universal Resource Locator): El dominio es un nombre único que identifica a un Sitio web.

Sitio Web: Un conjunto de páginas web forman un sitio web.
Para publicar un sitio web, se requiere del servicio de web hosting que le ofrecen los proveedores de presencia internet.

Web Hosting: Este servicio consiste muy a grandes rasgos, en la "renta de un espacio" en una computadora que se encuentra conectada ininterrumpidamente al Internet (Servidor Web) con la finalidad de almacenar sus páginas para que puedan ser visitadas.

WWW (World Wide Web): El Web, WWW o World Wide Web es un sistema mundial de información sencillo y poderoso, constituido por documentos y ligas a otros documentos; para seguir una liga basta con hacer "click" sobre ella con el mouse; y de esta forma se "navega" a través de la información.

4.3 Bibliografía.

[1] Java Media Framework Home Page http://java.sun.com/products/java-media/jmf/.

[2] IBM Media Software Home Page.
http://www.software.ibm.com/net.media/.

[3] Java Sound Home Page. Diciembre 2000.
http://java.sun.com/products/java-media/sound/index.html.

[4] Página sobre Player http://java.sun.com/marketing/collateral/jmf.html.

[5] Java 2. Curso de Programación. Fco. Javier Ceballos. Edi. Ra-Ma.

[6] Gifmanía. http://gifmanía.com/ y http://www.gifmania.com/.

[7] Java TM (Un lenguaje de programación multiplataforma para Internet)
Enrique Castillo, Angel Cobo, Patricia Gómez, Cristina Solares.

[8] http://atenea.udistrital.edu.co/estudiantes/omeza/resumen.html

[9] http://java.sun.com/products/java-media/jmf/

[10] Java Media Framework API Guide, 1999

[11]H. Schulzrinne et al., RFC 1889: RTP, A Transport Protocol for Real-Time Applications

[12]Franklin Kuo et al. Multimedia Communications, protocols and applicatiofns. Prentice Hall. 1998