VISUALIZADOR DE ANIMACIONES FLIC EN JAVA

Guillermo Aguirre

Proyecto UNSL-338403*

Departamento de Informática - Universidad Nacional de San Luis

Ejército de los Andes 950 - Local 106

5700 - San Luis - Argentina

Tel:(0652) 20823. e-mail: gaguirre@unsl.edu.ar

RESUMEN

Cada vez es más frecuente la aparición de animaciones en páginas Web y existe una gran variedad de alternativas para presentarlas. Algunas de estas alternativas requieren que sean abiertos una cantidad importante de archivos conteniendo los sucesivos cuadros que conforman la animación, pero es importante estimar el costo que se paga en tiempo de carga de la página debido a las conexiones http que son necesarias establecer para cada uno de los archivos abiertos. Una posibilidad que resulta atractiva en términos de minimizar los costos por accesos a archivos es la utilización de archivos con formato flic. Se requiere una única conexión http con el archivo que contiene, en forma compactada, el conjunto de imágenes que conforman la animación.

El procesamiento necesario para leer un archivo flic requiere interpretar adecuadamente el encabezado y cada uno de los distintos frames que conforman el archivo-animación, generando las sucesivas imágenes a partir de la definición de los pixels que vienen compactados en el archivo flic.

En este artículo se presenta la implementación en lenguaje Java de un visualizador de animaciones generadas en archivos flic y como son aprovechados los recursos del lenguaje para implementar el visualizador en sí y para la generación de las imágenes a partir de los bytes de datos que conforman la animación.

Palabras claves: Conexiones http, formato flic, generación de imágenes, lenguaje Java

* El grupo de investigación es subvencionado por la Univ.Nac. de San Luis y la ANPCyT (Agencia Nacional para la promoción de la Ciencia y la Tecnología)

1. INTRODUCCION

Son cada vez más numerosas las páginas web con animaciones y cuando son incluidas oportunamente permiten lograr presentaciones más atractivas.

Pero por otro lado las animaciones producen un aumento en la demora de carga de las páginas en que sean incluidas.

Uno de los factores que ha colaborado en el aumento del nivel de sofisticación de las páginas web ha sido la evolución que experimentaron los navegadores, pasando de los primeros navegadores orientados a texto como el www del CERN, a un escalón más arriba con entornos gráficos como el Mosaic y más arriba hoy en día en que cuentan con capacidades multimedia como el Netscape. Sin duda la posibilidad de ejecutar código escrito en lenguaje Java ha sido importante en esta evolución.

Los creadores de páginas buscan hacerlas lo más atractivas posibles, muchas veces combinando hipertexto con escenas en movimiento, pero existe otro factor también atractivo: la rapidez en la carga. Sin dudas lograr movimiento sin demoras seguramente es una buena combinación

Por lo general las animaciones pueden ser logradas de diferentes maneras: en base a una alta carga de procesamiento al construirse desde cero o bien son logradas mediante la apertura de varios archivos lo que afecta de manera directa el tiempo de carga. Esto último se produce en gran parte debido a la pobre sintonía que existe entre dos protocolos usados por word wide web para la transferencia de archivos el http y el tcp.

Por esto es importante tratar de minimizar el número de conexiones http para archivos que son parte de una página web.

Una alternativa para lograr presentar animaciones sin tener que pagar un alto costo en demoras de carga es el empleo de los archivos de formato flic ya que en un solo archivo de estos, se encuentra una animación completa y en forma compactada.

La sección dos presenta algunos aspectos generales de las animaciones. La sección tres aborda el tema de los protocolos empleados en World Wide Web y las razones por las que pueden originarse demoras debido a su utilización. La sección cuatro describe el formato de los archivos flic y el modo en que ha sido implementado el visualizador de animaciones.

2. CONSIDERACIONES GENERALES SOBRE ANIMACIONES

Todas las animaciones se basan en aparentar movimiento por medio de la presentación de sucesivos cuadros (frames) a una velocidad adecuada. Para lograr el efecto esperado se deben presentar entre 10 y 20 cuadros por segundo. El programa que muestra la animación debe tener un loop principal, que será el elemento coordinador de las diferentes tareas a realizar. Es su responsabilidad llevar el número de cuadro actual y solicitar periódicas actualizaciones de la pantalla. Este loop tendrá su propio thread[*] para correr ya que existen otras tareas que deben ser atendidas al mismo tiempo, como ser manejo de eventos y presentación de aquello que no es parte de la animación.

A los fines de implementar una animación se puede considerar una única clase que encapsula los datos necesarios y el conjunto de métodos a los que sabe responder para lograr presentar una animación.

La descripción de una clase genérica para reproducir cualquier tipo de animación sigue a continuación:

[*] Un thread es similar a un proceso real en el sentido que ambos son un único flujo secuencial de control. Pero un thread es considerado un proceso de peso liviano ya que corre en el contexto de un programa completo (full-blown) y toma ventaja de los recursos asignados a aquel proceso y el ambiente del programa

```
class Animacion extiende Canvas implements Runnable{
    • • •
    public void init(){
        //A partir del número de cuadros por segundo especificado por el usurio,
        //es determinada la demora entre cuadros y otras inicializaciones
    }
    public void start(){
        //Crear y comenzar el thread de animación
    }
    public void stop(){
        //Parar el thread de animación
    }
    public void run( ){
        //Bajar la prioridad del thread para que no interfiera con otras tareas
        //Registrar el tiempo de comienzo
        //Loop de animación
        while(/*el thread de animación esté corriendo/*){
            //Avanzar el número de cuadro
            //Mostrar el cuadro actual
            //Esperar el tiempo entre cuadros
        }
    }
```

El método que es invocado cada vez que es necesario actualizar la animación en la pantalla es *paint()*, el cual es invocado desde el método *run()*, que es el cuerpo principal del thread en que corre la animación y también es invocado automáticamente desde el AWT(Abstract Window Toolkit) cuando es necesario repintar algún componente[*]. El AWT es la parte del lenguaje Java que contiene un conjunto de clases para escribir programas con interfase de usuario gráfica.

```
Public void paint(){
    //dibuja el cuadro actual
}
```

La forma en que es implementado este método depende del tipo de animación que será empleada, las alternativas principales son: animar una serie de figuras geométricas, mover una imagen, o presentar una sucesión de imágenes. Este método trabaja en conjunto con *update()* para implementar *repaint()*. Estos tres métodos deben ser implementados en cada clase que extienda a la clase Componente si se requiere un comportamiento particular.

En relación a las variables de instancia, es necesario emplear al menos cuatro: una para llevar el número del cuadro actual, otra para una instancia de la clase Thread en la cual correrá la animación, una tercer variable con la demora en milisegundos entre los cuadros, y finalmente una de tipo boolean que indique si la animación está ejecutando o está parada.

Esta variable boolean será modificada en la implementación del método mouseDown que maneja la ocurrencia del evento de doble clic sobre el área de la animación. Ante la ocurrencia del evento el método detiene o reinicia la animación dependiendo del estado actual. Este comportamiento le da al usuario la posibilidad de detener la animación cuando lo crea conveniente.

[*] Instancia de la clase Componente o alguna de sus clases derivadas: Canvas, Frame, etc. Ver jerarquía de clases del paquete java.awt en Flanagan pág.281 [3]

```
public Boolean mouseDown(Event e, int x, int y){
        if(/* la animación está congelada */){
            //Invocar al método que inicia la animación
        } else {
            //Invocar al método que detiene la animación
        }
}
```

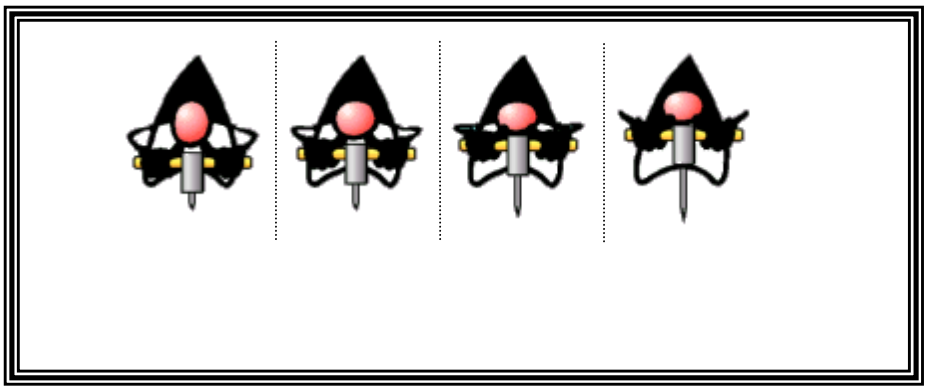
El pseudo código presentado puede ser adecuado para los diferentes estilos de animaciones que serán presentados a continuación, con algunas modificaciones en sus métodos.

Si la animación es implementada de manera que cada cuadro debe ser creado totalmente, como podría ser el caso de animar figuras geométricas, se corre el riesgo de que se produzca el efecto de parpadeo[*].

Un segundo tipo de animación consiste en mover una imagen sobre un background, el estilo Hollywood tradicional [4]. En este caso las dos imágenes ya han sido generadas de antemano: el background que permanece fijo y el objeto animado, denominado *cels* (en alución al celuliode sobre el cual se realizaban los dibujos para que su fondo fuera invisible). Una vez que las imágenes han sido cargadas, la animación consiste en presentar repetidamente al fondo en la posición fija establecida y al *cels* en la posición correspondiente, calculada según el número de frame a dibujar y el efecto buscado. Este cálculo puede demorar la presentación del cuadro por lo que se puede producir, aquí también, el efecto de parpadeo.

Un último tipo de animación es la secuencia de imágenes y consiste en presentar una imagen entera, que ya está generada, como un cuadro completo. El código para manejar este tipo de animación, es bastante simple ya que consiste básicamente en cargar una sola vez el conjunto de imágenes e ir presentando las sucesivas imágenes al ritmo establecido.

Pueden ocurrir dos tipos de inconvenientes: demoras en la carga de las imágenes, ya que cada una está en un archivo diferente, y como consecuencia de esa demora puede ser que se las muestre en forma parcial. Las soluciones para estos problemas podría ser por un lado comenzar a cargar las imágenes anticipadamente y por el otro, no iniciar la animación hasta que todas las imágenes hayan sido completamente cargadas, esto puede ser controlado usando la clase MediaTracker. Una alternativa para implementar este tipo de animaciones es utilizar una tira de imágenes:



A fin de minimizar el número de archivos abiertos con las imágenes requeridas, se puede utilizar un solo archivo que contenga una tira de imágenes. Para dibujar una imagen de la tira se establece el tamaño de una sola imagen y luego se recorta de la tira solo una parte, la

[*] El efecto de parpadeo se produce cuando se presentar sólo una parte de un cuadro en algún refresco de pantalla y recién en el próximo refresco se presenta el cuadro completo, el resultado es un breve corte en la animación.

correspondiente a una imagen.

Es posible hacer un análisis de los tipos de animaciones presentados en esta sección.

Tipo de animación	Costo de procesamiento	Archivos involucrados
Creadas desde cero	Alto, riesgo de parpadeo	Ninguno
Estilo Hollywood	No mucho, sólo para calcular la posición del cel	Uno para la imagen del fondo y otro para el cel
Secuencia de imágenes	Poco	Un archivo para cada imagen componente
Tira de imágenes	No mucho, sólo para extraer una imagen de la tira	Uno solo, cuyo tamaño es la suma de las imágenes componantes

Del cuadro anterior es posible concluir que la tira de imágenes es más conveniente si se consideran los parámetros mencionados.

Siguiendo la misma dirección con que fue introducida la tira de imágenes, la implementación que se describe más adelante aprovecha la simplicidad del código para manejar animaciones del tipo *secuencia de imágenes*, pero reduciendo al mímimo la cantidad de archivos con imágenes, ya que se utiliza un único archivo conteniendo todas las imágenes de la animación en forma compactada, este tipo de archivo son los flic, cuyo formato es detallado más adelante.

La próxima sección de alguna manera refuera la idea de mantener un número mínimo de archivos abiertos en cada página.

3. EL HYPERTEXT TRANSFER PROTOCOL (HTTP)

El protocolo de transferencia HTTP[7] es usado por World Wide Web para el intercambio de información entre servidores que se encuentran distribuidos.

HTTP necesita estar ubicado por encima de una capa que le provea de una corriente bidireccional de bytes, generalmente TCP. Cada interacción HTTP consiste del envío de un requerimiento por parte del cliente al servidor, seguido por una respuesta enviada del servidor al cliente. Tanto los requerimientos como las respuestas son creadas respetando los formatos establecidos y utilizando caracteres del formato ASCII simple.

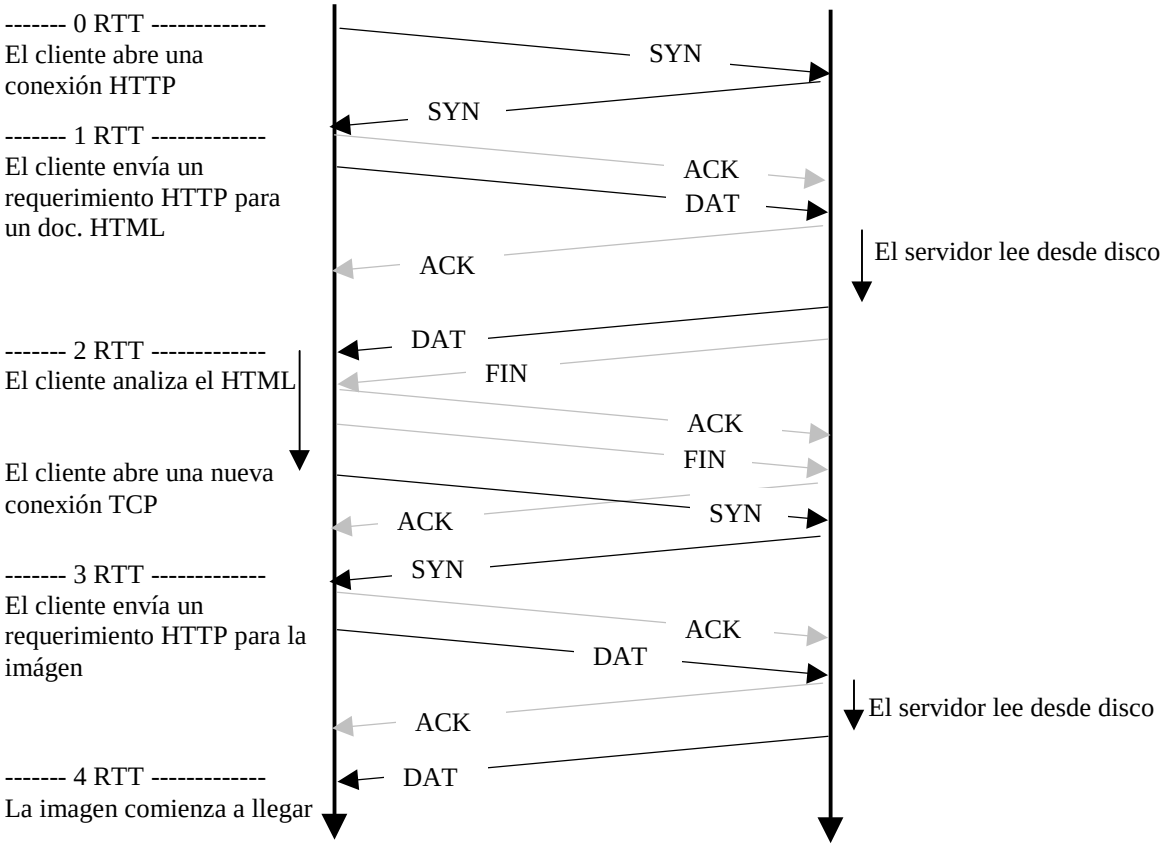
Los siguientes son los elementos que constituyen un requerimiento HTTP [6]:

a) Un método como ser GET para conseguir datos desde el URI especificado, PUT para almacenar datos en el URL especificado, LINK para enlazar un objeto existente con el objeto especificado.
b) Un Uniform Resource Identifier que es un string el cual identifica el objeto.
c) Un conjunto de encabezados de Hypertext Request donde se especifican cosas como ser el tipo de documento esperado, información de autenticación, etc
d) Un campo de datos opcional usado en métodos como PUT.

El server analiza el requerimiento y luego toma las acciones de acuerdo al método especificado. A continuación envía una respuesta al cliente en la cual se especifica:

a) Un código de estado que indica si el requerimiento tuvo éxito, y cuando no lo tuvo, la razones por las cuales falló.
b) Una serie de encabezados de objeto con información sobre el objeto retornado.
c) Opcionalmente un campo con el largo del contenido de la respuesta.

A continuación se describe la interacción HTTP entre clientes y servidores. La figura muestra los intercambios iniciales cuando se recupera un documento HTML que contiene una imágen. El tiempo se representa hacia abajo y las líneas diagonales representan paquetes que son enviados entre el cliente y el servidor. Los paquetes varian de acuerdo al tipo de paquete TCP que son intercambiados ha medida que progresa el diálogo.



Se puede ver en la figura los rounds trips times (RTT)[*] obligatorios que son originados por la combinación de los protocolos HTTP y TCP para una página HTML que contiene una imágen.

En el primer RTT el cliente abre la conexión TCP, resultando en el intercambio de paquetes como parte del proceso de diálogo que consiste en el intercambio de tres mensajes: un primer SYN, un segundo SYN, y un ACK final. Los paquetes SYN sincronizan los números de secuencia para las dos puntas de la conexión TCP, a fin de garantizar que nada se pierda o se cambie de orden.[8]

Durante el segundo RTT, el cliente transmite un requerimiento HTTP al servidor. El servidor después de leer de disco responde supongamos en un solo paquete y cierra la conexión TCP.

En el tercer RTT, después de analizar el documento HTML recibido, el cliente identifica el URL de la imagen que es necesario cargar e inicia otra conexión TCP con el intercambio SYN; SYN ACK.

Finalmente en el cuarto RTT, el cliente transmite el requerimiento HTTP ahora para la imagen y cuando esté listo el servidor comenzará a transmitir la imagen.

[*]RTT es el tiempo que insume un paquete en ir al otro extremo de la conexión y volver.

En síntesis para cada requerimiento de documento o imágen son necesarios dos network round trips, uno para abrir la conexión TCP y otro para pedir el documento o imágen. A esto se suma el hecho de que cada requerimiento HTTP solamente puede especificar un único objeto.

Si tenemos en cuanta que estas demoras se repiten para cada imágen de la animación, queda claro que en una página con animaciones una parte importante del tiempo de carga es debido a las conexiones HTTP que son necesarias establecer para cada archivo que contenga una imagen.

Otra deficiencia que puede ser remarcada es que aun cuando la conexión TCP puede estar activa sólo pocos segundos, la especificación del protocolo TCP requiere que el host que cerró la conexión recuerde cierta información de cada conexión por cuatro minutos despues de ser cerrarda. Esto puede llevar a tener las tablas de conexiones con muchas de sus entradas en estado "TIME-WAIT", lo que provoca el rechazo de nuevas conexiones o una costosa administración.

TCP permite que el extremo que envía comience a transmitir un nuevo segmento sin que se haya recibido el reconocimiento por un paquete anterior. La cantidad de datos que pueden ser enviados sin confirmación es administrada por el receptor y se la denomina tamaño de la ventana. Las implementaciones de TCP varian el tamaño de la ventana de acuerdo a una técnica que se denomina *slow-star* [5] por la cual se utiliza solo una parte del bandwhidth disponible de la red durante los primero round trips. El objetivo que se persigue mediante esta técnica es evitar congestion en la red. Slow-start consiste en ir aumentando gradualmente el tamaño de la ventana a medida que el extremo receptor lo permita. Esta aproximación es ideal para conexiones establecidas durante períodos relativamente largos de tiempo, pero en el caso de conexiones cortas, como las usadas en HTTP, el efecto de slow-start es contraproducente ya que no se aprovecha totalmente la capacidad de transferencia disponible.

4. ANIMACIONES DESDE UN ARCHIVO FLIC USANDO JAVA

En esta sección se presentan detalles de la implementación en Java de una clase que permite visualizar animaciones que han sido realizadas en archivos de formato flic. La primer subsección presenta detalles sobre el formato flic y el resto de esta sección, describe la implementación de los métodos principales que implementan el visualizador.

4.1. EL FORMATO FLIC

La propuesta original de este formato fué presentada por su autor, Jim Kent, en un paper del año 1993. Tomando una perspectiva general del formato, se percive como organizado en frames (cuadros) cada uno de los cuales contiene una imágen, opcionalmente una paleta de colores, y algunos datos complementarios.

Los primeros 128 bytes corresponden al encabezado que consiste en una serie de campos los cuales especifican, entre otras cosas, el tipo de archivo de animación (fli, flc o alguna variante), el tamaño del archivo y de las imágenes, número de frames, velocidad de reproducción, etc.

Una estructura en el estilo del lenguaje C para el encabezado es la siguiente:

```
typedef struct {
    DWORD size;          /*tamaño del flic incluyendo el encabezado*/
    WORD type;           /*tipo de archivo*/
    WORD frames;         /*numero de frames en el primer segmento/*
    WORD width;          /*ancho en pixels*/
    WORD height;         /*alto en pixels*/
    WORD depth;          /*bits por pixel*/
    WORD flags;          /*puesto en 0 ó 3*/
    DWORD speed;         /*demora entre frames*/
    WORD reserved1;      /*puesto en cero*/
    DWORD created;       /*fecha de creación*/
    DWORD creator;       /*número serial*/
    DWORD updated;       /*fecha de actualización*/
    DWORD updater;       /*número serial del actualizador*/
    WORD aspect_dx;      /*  */
    WORD aspect_dy;      /*  */
    BYTE reserved2[38];  /*puesto en cero*/
    DWORD oframe1;       /*offset del cuadro 1*/
    DWORD oframe2;       /*offset del cuadro 2*/
    BYTE reserved3[40];  /*puesto en cero*/
}
```

Para que los archivos sean reconocidos como válidos para la aplicación descripta más adelante, el valor del campo *type* debe ser 0xAF12 (archivos flc). La demora especificada en el campo *speed* está expresada en milisegundos. El campo *frames* contiene el total de cuadros del primer segmento sin contar el frame llamado *ring* que indican un salto al comienzo de la animación.

Todas las palabras y dobles palabras que constituyen un archivo flic están organizadas en la secuencia denominada Little Endian, el ordenamiento usado en los procesadores Intel de la línea 80X86. En este formato una doble palabra se almacena como:

Desplazamiento n	low
Desplazamiento n+1	high
Desplazamiento n+2	LOW
Desplazamiento n+3	HIGH

Desde una perspectiva más cercana, el formato está organizado como una jerarquía de chuncks. Los chunks de mayor jerarquía están compuestos por algunos de menor jerarquía, los subchunks. Todos los chunks tienen una parte fija (el encabezado) y una parte variante, la que depende del tipo de chunk. El encabezado de los chunks (16 bytes) contiene información sobre el tamaño completo del chunk, número de subchunks que lo forman,etc. Por su parte el encabezado de los subchunks (6 bytes) presenta dos campos: el tamaño y el tipo de subchunck.

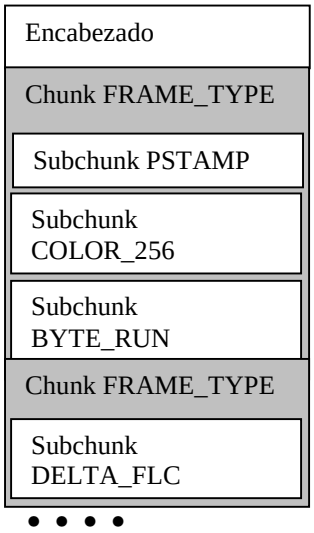
Los tipos de chunks posibles son:

Tipo de subchunks	Nombre	Descripción
4	COLOR_256	Paleta de colores de 256 niveles
7	DELTA_FLC	delta de imagen, compactada (archivos flc)
11	COLOR_64	Paleta de colores de 64 niveles
12	DELTA_FLI	delta de imagen, compactada(archivos fli)
13	BLACK	Frame negro
15	BYTE_RUN	Imagen completa compactada
16	FLY_COPY	Imagen sin comprimir
18	PSTAMP	Estampilla

Tipo de chunks	Nombre	Descripción
0xF100	PREFIX_TYPE	Chunk de prefijo
0xF1FA	FRAME_TYPE	Chunk de cuadro

Ejemplo de archivo flic

La siguiente es una posible disposición de un archivo flc típico:



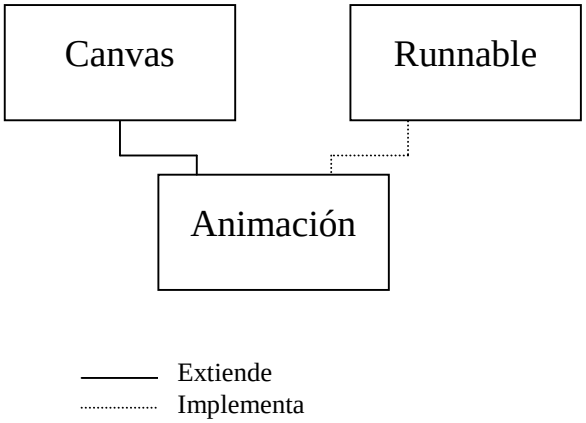
El primer frame está conformado por tres subchunks. El subchumk de tipo PSTAMP es una imagen reducida del primer frame. El de tipo COLOR_256 contiene una paleta de colores completa y está organizado en paquetes que contienen el número de entradas de la paleta anterior que no cambian o bien el número de entradas a cambiar seguido de los correpondientes tres byte RGB. De este modo se puede conformar la paleta actual. El último subchunk del primer frame contiene una imagen completa en forma compactada. Los BYTE_RUN están organizados por líneas, cada una de las cuales consiste de varios paquetes, los necesarios de acuerdo al ancho de la imagen. Cada paquete consiste de bytes de datos para copiar a la imagen, lo que se denomina un "literal run" o bien un sólo byte de dato que se repite en la imagen, conocido como "replicate run". Cada byte de datos debe ser transformado en el correspondiente pixel. El byte de dato es tomado como índice de acceso a la matriz que representa la paleta de colores actual, la primer columna contiene el byte correpondiente al rojo, la siguiente al verde y la tercera al azul. Cada uno de estos bytes debe ocupar su lugar correspondiente para formar el pixel.

Pixel=(255<<24) | (rojo<<16) | (verde<<8) | azul

El segundo frame y los que le siguen tienen un solo subchunk y es del tipo DELTA_FLC, la información contenida indica las diferencias que existen entre el frame actual y el anterior. También está organizado por líneas y dentro de cada una de ellas en paquetes los cuales pueden contener "literal run" o bien "replicate run". También puede haber un indicador de salto para indicar los pixels que no cambian.

4.2. DESCRIPCIÓN DE LA IMPLEMENTACIÓN

La implementación está concentrada en una sola clase:**Animación.** Para lograr cumplir con el comportamiento esperado, la clase **Animación** debe tener por un lado las capacidades normalmente esperables en objetos gráficos como la de poder agregarse en la ventana en que se visualiza el applet; poder cambiar su tamaño; poder volver a dibujarse cada vez que se lo soliciten, etc. y por otro lado la posibilidad de controlar el thread en el que correrá la animación. Por estos motivos es que **Animación** es declarada como una extensión de la clase **Canvas**, de la que hereda las capacidades gráficas, y además es una implementación de la interfase[*] **Runnable** para crear, comenzar, ejecutar y finalizar el thread donde se muestra la animación.



Jerarquía de clases

El resto de esta sección presenta el código de los métodos más importantes en recuadros sombreados con comentarios específicos sobre algunas líneas y el texto común son comentarios generales sobre los métodos

```
Public class Animación extends Canvas implements Runnable{
    protected int demora;              //demora entre imágenes
    protected Vector listaImagenes;    //lista de imágenes a ser mostradas
    protected Image imagenActual;      //imagen actual a ser mostrada
    protected Thread hebra;            //thread en que corre la animación
    protected int ancho, alto;         //tamaño de las imágenes
    protected int nroImagen;           //índice de la imagen actual en listaImagenes
    protected Boolean congelar;        // estado: corriendo o parada

    //algunas otras variables como la paleta y el arreglo de pixels
```

[*] Las unidades fundamentales de diseño en Java son los métodos públicos que pueden ser invocados para los objetos. Las *interfases* son una forma de declarar un tipo, consistente solamente de métodos abstractos y constantes, esto implica que los métodos deben ser implementados para cada aplicación particular de la interfase. Una clase puede implementar los métodos de una interfase de la manera que prefiera el diseñador de la clase. Así, una interfase tiene muchas más posibles implementaciones que una clase.

Constructor de la clase

```
public Animación(String nombreArchivo){
    demora=0;
    listaImagenes=new Vector();
    imagenActual=null;
    nroImagen=0;
    leeEncabezado(nombreArchivo); //para validar el archivo y leer ancabezado
}
```

El siguiente método es invocado para comenzar la animación

```
public void startAnimación(){
    if(hebra==null){
        hebra=new Thread(this);      //crea el thread para la animación
        hebra.start();               //asigna los recursos e invoca a run()
    }
}
```

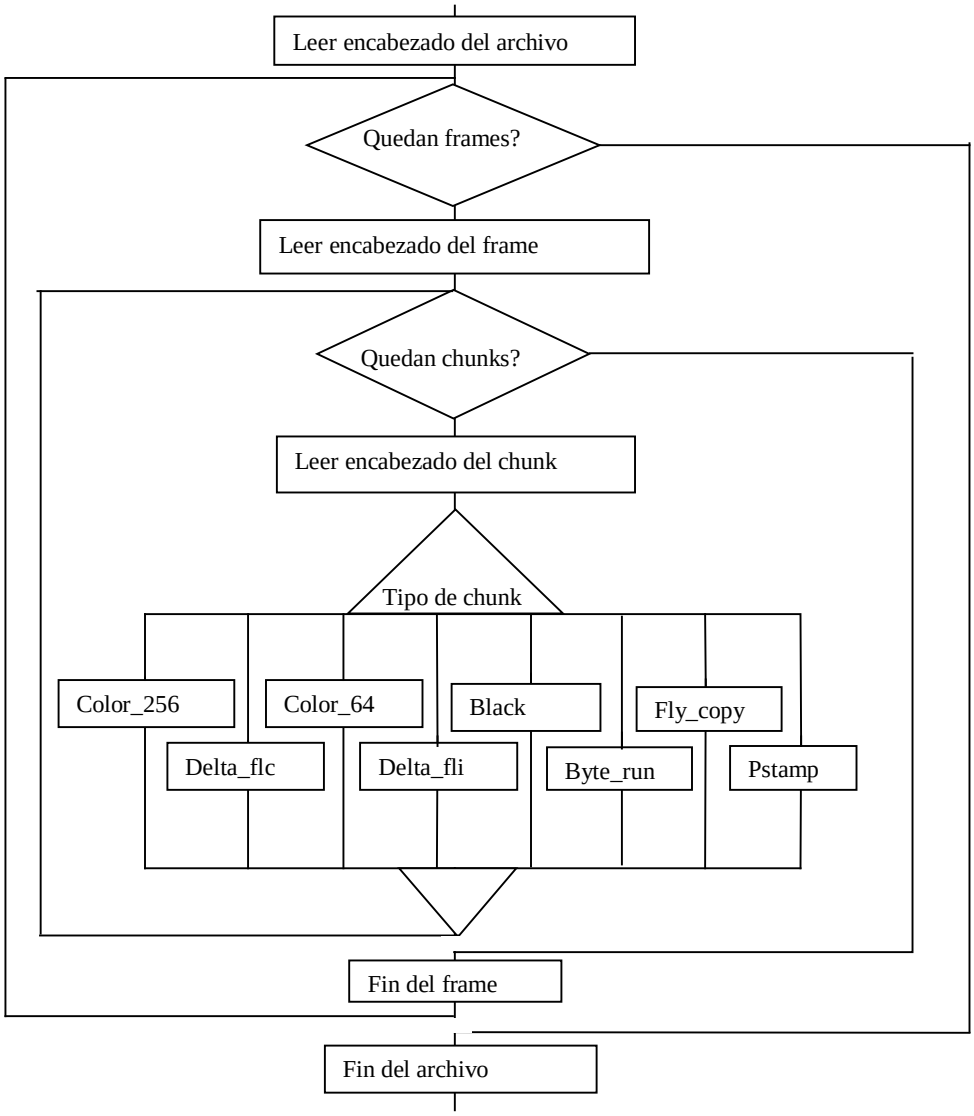
El método que sigue detiene la animación por pedido del usuario.

```
public void stopAnimación(){ //
    if(hebra!=null){
        hebra.stop();
        hebra=null;
    }
}
```

El método *run()* es invocado inmediatamente depues que el thread ha sido comenzado. Tiene como primer tarea generar la lista de imágenes luego de lo cual comienza el loop de animación.

```
public synchronized void run(){
    if(listaImagenes.size()==0)
        cargarListaImagenes();   //carga la lista de imágenes solo una vez
    while(hebra!=null){
        imagenActual = listaImagenes.elementAt(nroImagen);
        repaint();               //invoca a paint para actualizar la imagen
        try{
            wait(demora);    //demora entre cuadros
        }
            catch(InterruptedException e){}
        nroImagen=(nroImagen>listaImagenes.size()) ? 0 : nroImagen+1;
    }
}
```

A cerca de la generación de la lista de imágenes, no se presenta el código de impelementación por razones de claridad, pero si se describe a continuación su funcionamiento**.**



Para que el método privado **cargarListaImagenes()** pueda armar una lista con imágenes a partir de los datos que vienen compactados dentro del archivo, debe proceder frame por frame en secuencia. La lista de imágenes es conservada en una instancia de la clase *vector* que está predefinida en el lenguaje y que permite administrara los objetos que allí se mantengan. Es posible incorporar objetos de distintos tipos, en este caso imágenes, por medio de métodos predefinidos o bien recuperar el elemento que se encuentra en un lugar específico.

Solo son procesados los FRAMES_TYPE, ya que los PREFIX_TYPE son ignorados. Cada frame se comienza a procesar por el encabezado a fin de determinar el número de subchunks y su tamaño. Cada tipo de subchunks que es encontrado a medida que se avanza en el archivo se procesa del modo que corresponde a su tipo:

⇒Los subchunks que sean COLOR_256 y COLOR_64 implican una actualización de la paleta de colores. La matríz que representa la paleta debe ser actualizada en cada una de sus filas de acuerdo a la información contenida en el chunck. La paleta que resulte una vez procesado todo el chunk es la paleta que se empleará para las próximas imágenes.

⇒Los subchunks que sean DELTA_FLC, BYTE_RUN Y FLI_COPY implican la generación de una nueva imagen. Cada uno de estos tres tipo es implementado de manera diferente porque la

información que contienen necesita interpretación particular. Pero el resultado de su procesamiento es el mismo, generan una nueva imagen. Para lograrlo se crea un productor de imágenes que es una instancia de la clase MemoryImageSource:

ImageProducer productor = new MemoryImageSource(ancho, alto, pixels, 0 , ancho);

Donde el tercer argumento es la dirección de un arreglo con los pixels de la imagen que se quiere producir, comenzando desde la fila superior de la imagen y desde la izquierda hacia la derecha. Cada imagen para poder ser generada debe tener asociado su correspondiente arreglo de pixels. Luego el productor de imágenes *productor* es usado como argumento del método createImage que es heredado por la clase Canvas y es el que efectivamente crea la imágen.

Image imagen=createImage(productor);

El objeto *imagen* es incorporado como último elemento de la lista de imágenes.

⇒Los subchunks que sean BLACK generan una imagen como en el caso anterior pero cada pixel tiene el valor 0xF000 o sea negro.

⇒Los subchunks que sean PSTAMP son ignorados, podrían usarse para mostrar el primer cuadro de la animación antes de que comience a ejecutarse.

El método *paint()* es el encargado de presentar la imagen corriente:

```
public void paint(Graphics g){
    if(imagenActual!=null)
        g.drawImage(imagenActual, 0, 0, this);
}
```

El método *update()* modifica el comportamiento default ya que no necesita pintar el componente con el color del background porque la nueva imagen cubrirá completamente la anterior.

```
public void update(Graphics g){
    paint(g);
}
```

Los métodos **preferedSize()** y **minimunSize()** deben ser implementados para que AWT pueda dibujar el componente sin inconvenientes. Ambos retornan las dimensiones de las imágenes de la animación expresadas en pixels.

El método **mouseDown()** maneja los eventos de doble clic modificando la variable **congelar** y parando o recomenzando la animación.

5. CONCLUSIONES

Se ha presentado una variante económica para la inclusión de animaciones en páginas web.

La descripción de la interacción de los protocolos HTTP y TCP presentada aquí justifica sobradamante la necesidad de procurar diseños de páginas que incluyan la menor cantidad posible de archivos y la manera que los archivos con formato flic son una posbilidad conveniente para incluir animaciones sin pagar un costo demasiado alto.

Haciendo uso de las capacidades para la administración de thread y para la creación de imágenes provistas por el lenguaje Java se presentó el diseño y parte de la implementación de una

clase que permite leer un archivo con formato flic a partir del cual se recupera el conjunto de imágenes que lo conforman para posteriormente proceder a reproducir la animación.

6. BIBLIOGRAFÍA

[1] The Java Tutorial. Object-Oriented Programming for the Internet Mary Campione, Kathy Walrath Addison Wesley.

[2] The Java Programming Languaje Ken Arnold, James Gosling Addison Wesley

[3] Java in the nutshell D. Flanagan O´Reilly & Associates.

[4] Manual de Autodesk Animator Studio Autodesk

[5] Analysis of HTTP Performance problems Simon Spero

[6] Improving HTTP Latency Venkata Padmanabhan y Jeffry Mogul

[7] Hypertext Transfer Protocol Tim Berners-Lee. Este es un working draft.

[8] TCP/IP Architecture, Protocols and Implementation Sidnie Feit (McGraw-Hill)