CONTENIDO

1. Descripción del proyecto.

En procesos de tele-{control, adquisición, medida, ...} es necesaria la transmisión de imágenes (tomadas en un punto lejano) hasta el operador, usando un canal de comunicación que puede presentar anchos de bandas bajos y variables en el tiempo. Ejemplos típicos pueden ser la toma de imágenes astronómicas, donde ocurre que el astrónomos puede estar usando un telescopio que se encuentra alejado geográficamente, o la toma de imágenes microscópicas de especímenes biológicos, donde algo semejante puede ocurrir con el biólogo. En estas situaciones, una aplicación que permita la visualización lo mas rápida posible de la imagen (en general de grandes dimensiones) o de una sección de esta, puede ser muy útil.

El televisualizador debe ser una aplicación que corre en dos computadoras remotas. Una de ellas, a la que llamaremos servidor, almacena en disco la(s) imagen(es) que se desea visualizar desde la otra computadora, a la que llamaremos cliente. Se trata por tanto de dos procesos independientes. El primero corre en el servidor (proceso servidor) y su tarea consiste en, recibida una petición por parte de una computadora cliente (que corre uno o mas procesos clientes), leer una imagen del disco (que debe estar debidamente comprimida) y enviarla a través del canal de comunicación entre ambas computadoras (típicamente Internet). El proceso cliente tiene la misión de realizar una presentación progresiva de la imagen de la forma mas eficientemente posible (desde el punto de vista visual), en cuanto suficientes datos nuevos hayan sido recibidos.

3. Especificación del problema.

Actualmente, si un astrónomo quisiera tomar imágenes a través de un observatorio de unas cualidades mínimamente aceptables, para poder luego visualizarlas y realizar cualquier tipo de calculo sobre ellas, tendría solamente dos opciones, la cual mas costosa: tendría que desplazarse al observatorio en cuestión y tomar dichas imágenes, tratándolas allí mismo o en cualquier otro sitio que el considerase, siempre y cuando tuviese algún medio para transportar las imágenes. Las imágenes astronómicas suelen ser de grandes dimensiones y ocupar gran cantidad de memoria, por lo que transportarlas de un sitio a otro puede resultar un serio inconveniente. La otra opción es que alguien tome esas imágenes y las ponga a disposición del astrónomo mediante algún canal de comunicación (como pueda ser Internet). Igual de engorrosa resulta esta ultima opción, ya que para poder transmitir una imagen de grandes dimensiones a velocidades aceptables, se requiere un canal de comunicación con un ancho de banda elevado, careciendo Internet de esta cualidad en la mayoría de los casos. La segunda opción ha provocado el desarrollo de este proyecto que puede ser de gran ayuda para los astrónomos que trabajan habitualmente en el observatorio de Calar Alto, situado en la sierra de Almería.

Lo que se requiere es almacenar las imágenes que se toman del telescopio en una maquina servidora, de manera que cualquiera pueda conectarse a ella a través de Internet y visualizar en su computadora cualquiera de las imágenes almacenadas.

La maquina servidora de imágenes es una maquina Sun, con un sistema operativo con arquitectura UNIX (Solaris) y con una velocidad de procesamiento elevada. La velocidad de su conexión a Internet también es modesta (2 Mbps). Sin embargo, la conexión de los clientes a la red suele ser de mucha menos velocidad (en torno a 24 Kbps).

4. Estructura de la aplicación a desarrollar.

Evidentemente, la aplicación en si consta de dos partes diferenciadas: un programa servidor, y un programa cliente.

El programa servidor correrá en la maquina UNIX y se encargara de atender peticiones que le lleguen a través de Internet. Las conexiones que los clientes hagan con el servidor serán solo de una petición, por razones que se comentaran mas adelante, es decir, que por cada conexión, el servidor atenderá una sola petición, cerrando posteriormente la conexión. Si son necesarias varias peticiones por parte del cliente, este establecerá tantas conexiones como precise. Quizás esto sea un factor negativo, en cuanto a perdida de tiempo tratando de establecer las conexiones, pero, como se podrá comprobar mas adelante, se gana en flexibilidad y fiabilidad. Las peticiones serán de tres tipos diferentes: mandar una imagen, mandar un directorio entero de imágenes (mandara todas las imágenes que se encuentren en el directorio seleccionado por el cliente) o mandar la jerarquía de directorios disponibles en la maquina del servidor, para que el cliente pueda ver a que directorios de imágenes puede acceder. El servidor se lanzará en el directorio a partir del cual colgarán distintos subdirectorios con imágenes almacenadas, aunque también sea posible almacenar imágenes en el mismo directorio que el servidor. Debe además impedir el acceso de los clientes a directorios que no sean el mismo que el del servidor o descendientes del mismo, por cuestiones de seguridad.

El programa cliente deberá de ser capaz de mostrar en pantalla la imagen seleccionada, cuando la petición del usuario sea la de obtener una imagen. Cuando la petición sea la de ver la jerarquía de directorios debe mostrar la jerarquía gráficamente. Y cuando la petición sea la de obtener todas las imágenes de un directorio, estas deben mostrarse ante el usuario en forma de mosaico, junto con su nombre.

La arquitectura sobre la cual correrá el programa servidor esta bien definida, y no admite ninguna variación. En cambio, la arquitectura sobre la cual correrá el programa cliente no es fija, pudiendo variar según el cliente y la máquina que posea, así como el sistema operativo que emplee.

5. Estudio y defensa del conjunto de alternativas adoptadas.

5.1. Algoritmo de compresión SPIHT.

Las imágenes que se almacenen en la máquina servidora deberán estar en un formato de compresión determinado. Los formatos actuales mas óptimos de compresión y descompresión de imágenes, tales como JPEG, GIF o PNG, no nos permiten llevar a cabo la función deseada. Es cierto que consiguen ratios de compresión difícilmente superables, y que la mayoría de ellos permiten la visualización por filas (o columnas) de la imagen conforme se va recibiendo nuevos datos. Como desventaja añadida tenemos que un observador no puede distinguir medianamente el objeto retratado en la imagen hasta no haber recibido la casi totalidad de la misma. Es necesario por tanto un algoritmo de compresión-descompresión que se beneficie al máximo de las cualidades de las imágenes con las que vamos a trabajar.

En el IEEE Transactions on Circuits and Systems for Video Technology, Volumen 6 de Junio de 1996, Amir Said y William A. Pearlman exponen un nuevo algoritmo de codificación de imágenes llamado SPIHT (Set Partitioning in Hierarchical Trees). No está entre los objetivos de este proyecto explicar el funcionamiento del mismo, sino exponer las cualidades que llevaron a elegir este algoritmo para su implementación e incorporación a la solución adoptada para el proyecto.

SPIHT nos permite mostrar la imagen completa a partir del primer bit recibido, de manera que el observador podrá ver como la imagen pasa de ser paulatinamente un cuadro grisáceo a una imagen perfectamente distinguible en un tiempo considerablemente corto, y con una cantidad de información recibida escasa. En la mayoría de los casos, con sólo transmitir un 1% de la imagen, ya comprimida, el cliente u observador podrá ver ya la imagen perfectamente, quedando ya a su disposición el parar su visualización o esperar a que se transmita por completo para verla con total nitidez.

A continuación se muestran una serie de frames, todos relativos a la misma imagen, capturados en diferentes momentos de una visualización:

Como se puede apreciar, desde un primer momento podemos observar algo, aunque carezca de total claridad, pero transmitiendo al observador una sensación de velocidad en el proceso. Con el 1% aprox. de la imagen (en algunos casos, incluso un porcentaje mas bajo) podemos ya distinguir prácticamente por completo la escena representada en la imagen. Además, si se quisiera mostrar la imagen escalada a bajas dimensiones, los defectos propios de los primeros frames del proceso de visualización progresiva estarían menos acentuados. Imaginemos que tenemos 100 imágenes del mismo tamaño, pues en el mismo tiempo que tardaríamos en bajarnos completamente solo una de ellas, podríamos bajarnos el 1% de cada una de las cien y poder observarlas.

El algoritmo, además, es fácil de implementar, siendo el tiempo dedicado al mismo relativamente corto. Su velocidad de ejecución, independientemente la arquitectura y lenguaje utilizado para su implementación y posterior puesta en marcha, es bastante alta.

Cuando hablamos de velocidad de ejecución, hemos de fijar nuestra atención solamente en el descompresor por razones obvias: el compresor apenas tiene relevancia en este sentido, ya que las imágenes se guardan ya comprimidas, no han de comprimirse en cada transmisión, sino que se comprime una sola vez por cada imagen. Por eso, la velocidad del compresor no es un factor determinante. Sin embargo, el descompresor si adquiere en este aspecto un papel relevante, al realizarse la operación de descompresión cada vez que se quiera visualizar una imagen. Es por eso, que la velocidad del descompresor debe de ser medianamente aceptable, de lo contrario no se podría conseguir una visualización mas rápida de la que ya la hacen los actuales estándares de compresión de imágenes. Aunque se haya dicho que el algoritmo en cuestión permite implementaciones con unas ejecuciones rápidas, es necesario comentar que para la visualización de la imagen o frame, es necesario aplicar una transformada S+P inversa bidimensional al producto de dicho proceso, cálculo que consume un tiempo considerable de cualquier procesador, y más si la imagen es de grandes dimensiones.

El algoritmo de compresión, que se puede encontrar en ingles en la documentación antes mencionada, es el siguiente:

1. ***Inicialización:*** *almacenar n = [log 2(max(i,j){|ci,j|})]; incializar LSP como una lista vacía y añadir las coordenadas (i,j) ∈* ***H*** *a la LIP, y solamente aquellas de las que desciende a LIS, como entradas de tipo A.*

2. ***Paso de ordenación***

2.1. Para cada entrada (i,j) en el LIP hacer

2.1.1. Almacenar Sn(i,j)

2.1.2. Si Sn(i,j) = 1 entonces mover (i,j) a la LSP y almacenar el signo de Ci,j

2.2. para cada entrada (i,j) en el LIS hacer:

2.2.1.si la entrada es del tipo A entonces

· *Almacenar Sn(D(i,j));*

· *Si Sn(D(i,j)) = 1 entonces*

** Para cada (k,l) ∈* ***O****(i,j) hacer:*

- Almacenar Sn(k,l):

- Si Sn(k,l) = 1 entonces añadir (k,l) a la LSP y almacenar el signo de Ci,j

- Si Sn(k,l) = añadir (k,l) al final de LIP

** Si L(i,j) < > 0 mover (i,j) al final de LIS, como una entrada de tipo B, e ir al paso 2.2.2.; sino; eliminar la entrada (i,j) de la LIS*

2.2.2. Si la entrada es de tipo B entonces:

· *Almacenar Sn(L(i,j));*

· *Si Sn(L(i,j)) = 1 entonces*

** Añadir cada (k,l) ∈* ***O****(i,j) al final del LIS como una entrada de tipo A*

** Eliminar (i,j) de LIS*

3. ***Paso de refinamiento:*** *para cada entrada (i,j) en la LSP, excepto aquellas incluidas en el ultimo paso de ordenación, almacenar el n-esimo mas significativo bit de |Ci,j|;*

*4.**Paso de cuantizacion**: decrementar n en 1 e ir al paso 2.*

(Para información extra acerca del algoritmo, remítase la documentación aportada al respecto. Sólo se muestra aquí para poder ver el grado de complejidad que posee).

Este algoritmo almacenaría el resultado de la compresión en una archivo determinado. El descompresor es prácticamente igual, salvo que en vez de almacenar, lee.

Es pues, el algoritmo SPIHT de compresión-descompresión de imágenes el adoptado para el proyecto, descartando los actualmente vigentes en Internet (GIF, JPEG, PNG, TIFF, etc.), por no ofrecer la cualidades necesarias.

5.2. C como lenguaje base para el servidor.

Elegir C para realizar cualquier programa, es en muchos casos la mejor opción, por su de sobra conocida, potencia. Pero en este caso, además de la potencia tenemos a nuestro favor el robusto soporte que ofrecen los sistemas operativos con arquitectura UNIX, como es nuestro caso, a este lenguaje (de hecho, la mayor parte de estos sistemas operativos están escritos en este lenguaje). Debido a que vamos a trabajar a nivel de comunicaciones, archivos y directorios, nos resulta imposible crear un código portable a cualquier plataforma. No obstante, esto no es muy agravante, ya que de momento y por un largo tiempo la maquina servidora no tiene vistas a variar. Si esta variase, y no tuviese el mismo sistema operativo, no seria extremadamente difícil reescribir el código del programa servidor, o incluso hacerlo en otro lenguaje, ya que este no es nada complejo.

Para el caso concreto que aquí se trata, el lenguaje C es la mejor opción, por encima de cualquier otro lenguaje que actualmente se emplea. Aunque partiésemos de la hipótesis de que no conocer este lenguaje de antemano, ni de programación con sockets, ni de manejo de archivos ni directorios, sería mas fácil encontrar ejemplos, en los cuales basarnos, para este lenguaje y el sistema operativo elegido, que para cualquier otro lenguaje y/u otro sistema operativo.

5.3. Java como lenguaje base para el cliente.

Las razones de elegir Java como lenguaje base para la implementación del programa cliente no son tan obvias como pasaba con el programa servidor.

De todos es sabido la fantástica cualidad que ofrecían los creadores de Java a los desarrolladores: en la teoría menos empírica, cualquier programa creado con Java podrá ejecutarse en cualquier computadora que posea cualquier sistema operativo, siempre y cuando se posea una maquina virtual adecuada. Esto, en la practica resulta no ser tan directo, observando que, a medida que el programa crece en complejidad y envergadura, se hace cada vez mas difícil que el mismo programa se comporte de forma idéntica en cualquier arquitectura.

Java es un lenguaje compilado, pero también interpretado, es decir, el código fuente se compila para producir un código de una maquina virtual (la maquina virtual Java), también llamado byte-code (tal vez porque cada instrucción ocupa solo 8 bits, un byte). Este byte-code es luego interpretado por la maquina virtual, que no es más que otro programa, este ya sí, dependiente de la arquitectura. Surge pues la idea de que, visto lo visto, al ser medio compilado y medio interpretado, no ofrecerá la misma velocidad que un programa compilado y con código de maquina nativo. En efecto, la velocidad de un programa desarrollado en Java, en el mejor de los casos, se acerca al 80% de la velocidad de un programa con código nativo, y el peor de los casos llega a tener una lentitud desesperante. Influye también en gran medida la maquina virtual que se posea.

No obstante, existen muchas razones por la que decantarse por Java: es un lenguaje moderno, con mucho auge actualmente, y que tardará mucho tiempo en desaparecer (si es que desaparece). Es mas, su cualidad de “independiente de arquitectura” permite su integración en Internet, en las paginas Web, además de que no es necesario tener distintas versiones del mismo programa para poder abarcar muchas plataformas diferentes. En cuanto al tema de la velocidad, actualmente todos los navegadores de Internet mas modernos y famosos, como puedan ser Internet Explorer o Netscape Navigator, incorporan ya el mecanismo JIT (Just In Time). Este mecanismo permite que, en vez de interpretar el código Java, se compile y se genere código maquina, produciendo de esta manera un programa con una velocidad de ejecución comparable a la que proporcionaría cualquier otro lenguaje.

Es cierto que en muchas ocasiones, un programa escrito en Java se vuelve dependiente de la arquitectura, ya no sólo en su ejecución, que depende en cierta medida de la maquina virtual, sino también en la interfaz de usuario. En los JDKs (Java Development Kit) anteriores a la versión 1.2.2, o Java 2, y con los paquetes de la AWT, se podían hacer interfaces de usuario rápidamente, aunque gráficamente pobres. Estas eran muy dependientes del entorno gráfico en la que se ejecutaban, de manera que un botón podría no tener la misma apariencia en Windows 98, que en X-Window para Linux con Gnome. Sun corrigió esto con la creación de las JFCs (Java Foundation Classes), dentro de las cuales podemos encontrar los paquetes de la Swing, componentes gráficos con lo que construir una interfaz de usuario que, además ser mucho mas eficiente que la realizada con AWT, tendremos la seguridad de que tendrá el mismo aspecto independiente del entorno gráfico en el que se ejecute.

Para nuestro caso en concreto, el cual, el cliente estará desarrollado en Java compatible con la versión 1.1.5 o superior, no será necesario emplear las clases Swing, ya que no tendremos componentes gráficos complejos, y así nos aseguraremos un abanico mucho mas amplio de clientes potenciales, ya que las JFCs no las soportan a día de hoy por defecto todos los navegadores.

Como valor añadido, cabe decir que el lenguaje de Java es mas fácil y rápido de aprender que muchos lenguajes, además de que desarrollar una aplicación con el requiere de un menor tiempo debido a que su diseño evita errores de programación muy comunes (mal uso de punteros, mala administración de memoria, etc.). Finalmente el código fuente suele ser bastante más legible que, por ejemplo, un programa desarrollado con C, que en el pero de los casos, no hay persona capaz de adivinar su funcionalidad.

6. Estudio de viabilidad

Para el desarrollo del proyecto no es necesario un estudio de viabilidad, puesto que todos los elementos implicados en el mismo son totalmente accesibles, sin coste alguno.

Compiladores, entornos de desarrollo, documentación, instalación y puesta en marcha, son elementos o procesos totalmente accesibles, sin requerir nada extra.

En la mayoría de los entornos UNIX viene ya por defecto un compilador, junto con un manual donde podemos consultar las diferentes llamadas al API del sistema operativo.

El JDK (Java Development Kit) está disponible gratuitamente en Internet, en el sitio web de Sun, tanto sus versiones más antiguas como la más moderna (que en el momento de escribir este documento es la 1.3).

La documentación acerca del algoritmo de compresión de imágenes llamado SPIHT esta totalmente disponible, también sin coste alguno, en la red Internet.

Tenemos pues una viabilidad total a la hora de desarrollar el proyecto.

7. Estimación temporal de las tareas a realizar

Mas que un estimación temporal, es un recuento de las horas empleadas para la realización del proyecto, dividido en dos grandes apartados, el desarrollo del programa servidor y el programa cliente escrito en Java.

El recuento de horas es el siguiente:

Elemento	Lenguaje	Horas
Programa servidor	C	48
Programa cliente		
Applet principal	Java	96
Componentes gráficos	Java	28
Descompresión SPIHT	Java	12
Clase para la carga de imagen	Java	48
Clase para la carga de directorios	Java	54
Clase para la carga de la jerarquía	Java	22

Tenemos un total de 308 horas (en el tiempo de cada tarea se ha incluido, no solo el tiempo de implementación, sino también el tiempo de prueba y verificación del elemento relacionado).

8. Plan de pruebas

A la hora de realizar las pruebas, toma una mayor importancia el programa cliente que el servidor, ya que es necesario comprobar su velocidad de ejecución en distintas plataformas y navegadores.

La primera prueba fue corriendo el servidor en la misma máquina que el cliente. Se empleo el sistema operativo Linux en su distribución Red Hat 7.0, y el navegador Netscape Navigator 4.7. La velocidad que demostró el cliente escrito en Java era bastante lenta.

Una segunda prueba, realizada ya a través de la red, corriendo el servidor en una máquina UNIX, a la cual se conectaba el cliente, ejecutándose este en una máquina situada justo a lado de la del servidor, empleaba también el Netscape Navigator 4.7, pero con la diferencia de que el sistema operativo bajo el que iba a correr el cliente sería el Ms-Windows 98. Cual fue la sorpresa cuando el cliente demostró una velocidad de vértigo, como si fuese un programa con código nativo. Igual fue la velocidad que demostró usando el navegador Internet Explorer 5.0.

La velocidad de visualización de la imagen apenas variaba, en Windows, claro, al ejecutar el cliente desde una máquina remota con una conexión a Internet pésima y con un módem de 56kbs.

La diferencia entre Ms-Windows 98 y Linux Red Hat 7.0, en cuanto a velocidad del cliente, no tiene nada que ver con el sistema operativo, sino con los navegadores. Aunque se empleo la misma versión de Netscape Navigator tanto en un sistema como en otro, la versión para Windows tenía incorporado el mecanismo JIT, que compila el código Java y produce código nativo, al igual que el Internet Explorer, pero por el contrario, en la versión para Linux, esta opción no venía. De igual modo, en versiones superiores del Navigator para Linux, ya viene incorporarda esta opción.

9. Descripción detallada de la aplicación desarrollada.

9.1. Formato de almacenamiento de imagen.

Las imágenes se almacenaran directamente como salen después de aplicarles el compresor SPIHT, añadiéndoles al principio, como cabecera, la anchura y la altura de la imagen. Tanto la anchura como la altura irán almacenadas en dos bytes cada una, en el siguiente formato:

Byte más significativo	Byte menos significativo	Ancho
Byte más significativo	Byte menos significativo	Alto

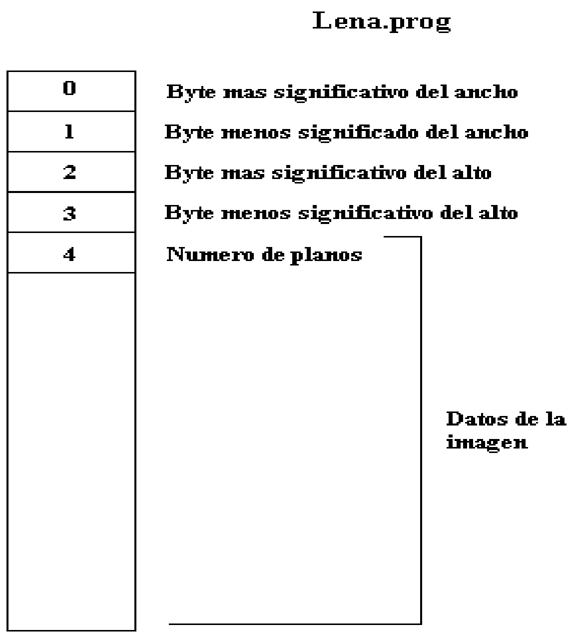
De esta manera, al transmitir la imagen byte a byte, el primer byte que le llega al cliente es el byte más significativo de la anchura, y el segundo es el byte menos significativos. El tercero será el byte más significativo de la altura y el cuarto el byte menos significativo.

El resto del archivo de la imagen se pasa directamente al descompresor conforme se vaya recibiendo.

El compresor de imágenes no será descrito en esta documentación, ya que apenas posee importancia, siendo simplemente la implementación del algoritmo del SPIHT, que ya viene en pseudo-código en la documentación correspondiente.

Las imágenes, por convenio, se guardarán con extensión .prog, para poder diferenciarlas en el servidor, puesto que esto para el cliente es transparente.

A continuación se muestra una figura con información acerca de una imagen .prog:



9.2. Protocolo de comunicación cliente-servidor.

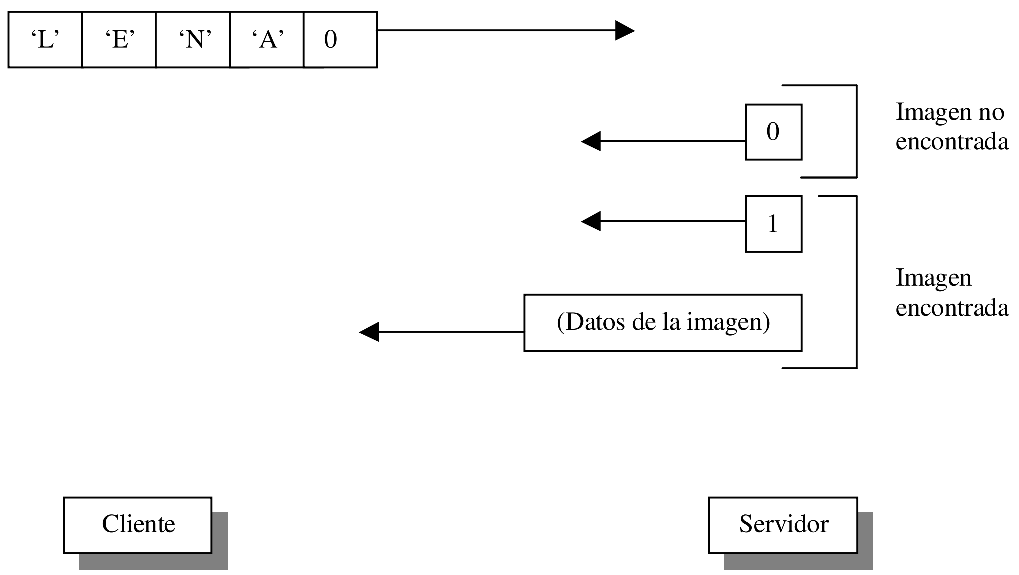
El servidor solo acepta una petición por conexión. Solo existen tres tipos de peticiones por parte de los clientes:

· Petición de imagen:

El cliente mandara al servidor el nombre de la imagen que quiera recibir, mas un byte a cero, para indicar el final de la cadena, del mismo modo a como se almacenan las cadenas en C. El servidor buscará la imagen solicitada y en el caso de que no la encuentre, mandará un byte al cliente con el valor cero. En caso de que sí la encuentre, le mandará al cliente un byte con el valor 1 y seguidamente el archivo de la imagen solicitada. Después, cerrará la conexión.

El nombre de la imagen no podrá llevar la extensión, ya que se la añade el servidor automáticamente. También puede contener una ruta, pero siempre relativa al directorio propio donde se esta ejecutando el servidor. No son válidas las rutas que contengan el carácter '.', por seguridad. El nombre de la imagen, contenga o no ruta, es añadido al directorio del servidor. De esta manera, si el servidor corre en /root/server/, y el cliente envía lena, el resultado a buscar en el servidor será /root/server/lena.prog.

Debajo se expone un breve esquema de la petición:



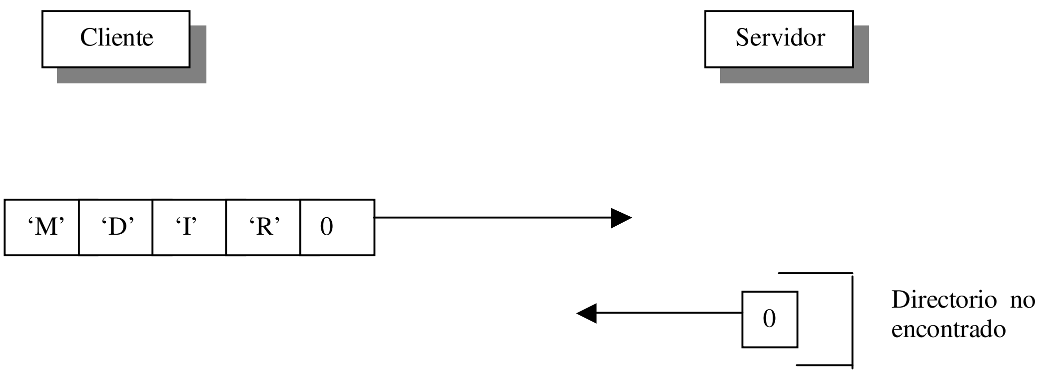
· Petición de un directorio:

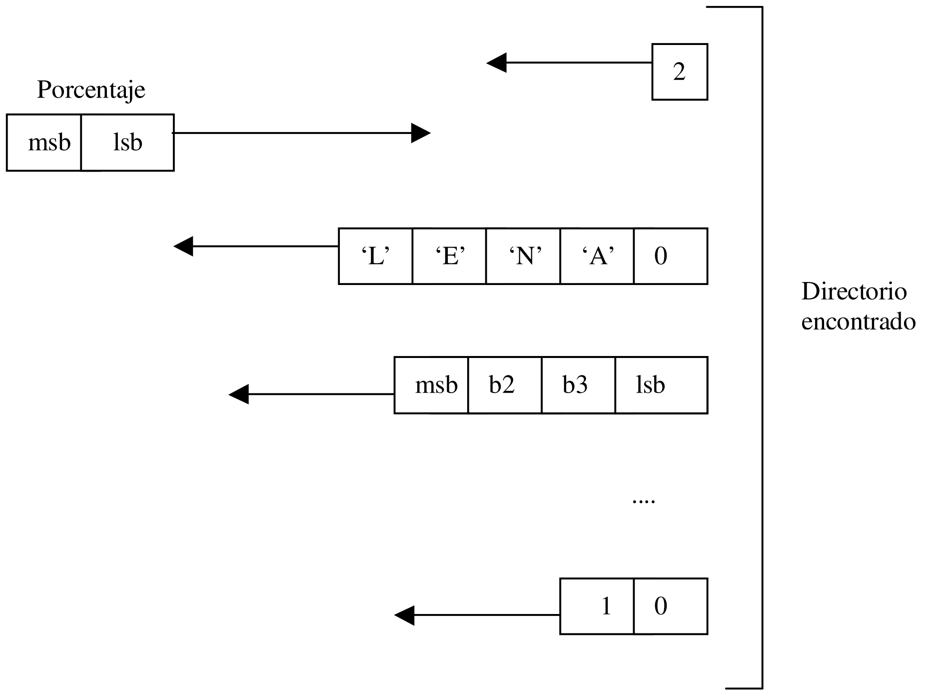
El mecanismo es similar al anterior. El cliente manda una cadena terminada en cero al servidor con el nombre del directorio a cargar. El servidor unirá este nombre a su directorio y comprobará que efectivamente se trata de un directorio válido, en cuyo caso mandará al cliente un byte con valor 2. En caso contrario enviará un cero. Si el cliente recibe una respuesta afirmativa por parte del servidor, deberá enviarle a este el porcentaje multiplicado por 100 de bytes a leer por cada imagen del directorio, almacenado en dos bytes. De esta manera, si el porcentaje está almacenado en dos bytes (una palabra de 16 bits), mandara primero el byte más significativos y después el byte menos significativo. El servidor pasa entonces a mandar el porcentaje de bytes de cada imagen almacenada en el directorio especificado. Pero, el cliente debe saber cuantos bytes va a recibir de cada imagen, para saber cuando se acaba una y empieza otra.

Para cada imagen a enviar, el servidor transmite, antes de los datos propios de la imagen, el nombre de la misma acabada en cero, conteniendo incluso la ruta relativa propia, y la cantidad en bytes va a transmitir de la misma. Esta cantidad va almacenada en una palabra de 32 bits y se transmite como de costumbre, primero el byte más significativo, hasta llegar al menos significativo.

Cuando el cliente recibe como nombre de imagen una cadena cuyo primer byte tiene el valor 1, sabrá que ya no quedan mas imágenes por cargar.

Debajo se expone un breve esquema de la petición:





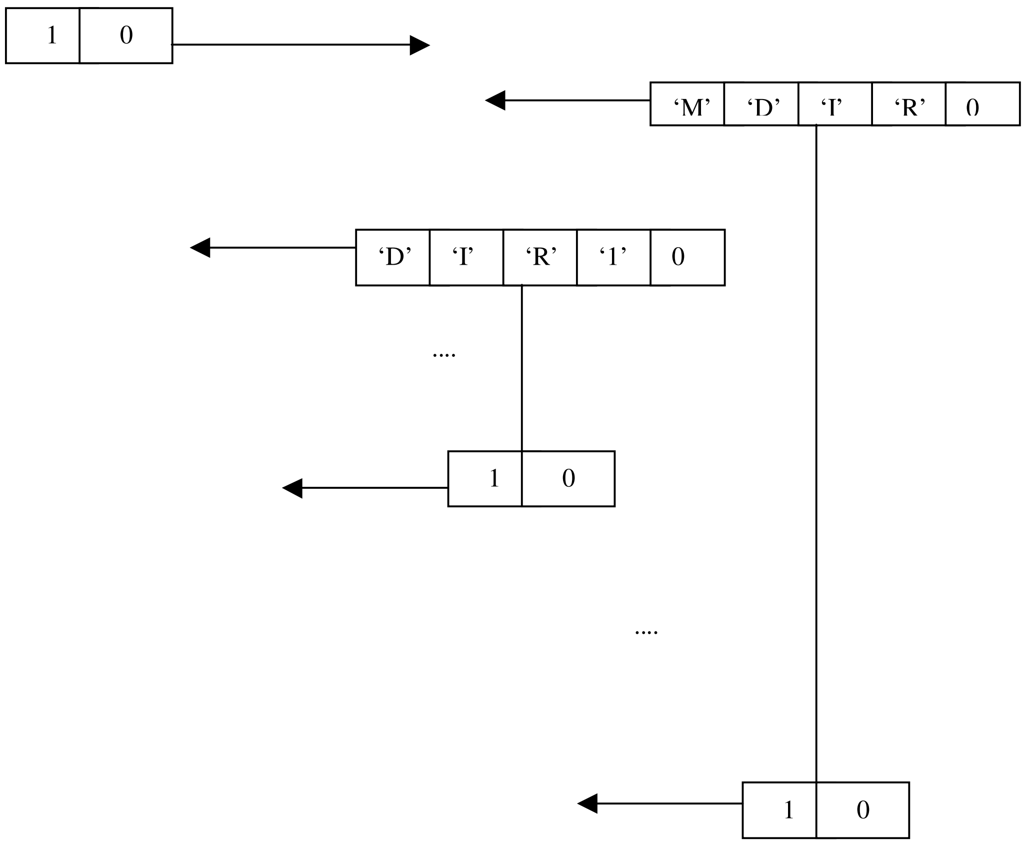
· Petición de la jerarquía de directorios:

El servidor sabrá que la petición que el cliente quiere realizar es la de obtener la jerarquía de directorios, cuando le llegue una cadena como nombre de imagen o directorio cuyo primer byte tenga el valor 1. Esto querrá decir que el servidor deberá de recorrer todos los subdirectorios de los cuales cuelga y mandar sus nombres al cliente.

La forma en que lo hace es recursiva, por lo que, para poder recibir correctamente los directorios es necesarios que se reciban también de forma recursiva.

El servidor comenzará con el directorio actual en el que se encuentra, buscando todos los directorios que posea. Cuando finalice de mirar en el directorio (o cualquier otro), mandará una cadena cuyo primer byte tiene el valor 1. Para cada directorio que encuentra, manda su nombre y busca dentro de él, siguiendo el mismo método.

Debajo se expone un breve esquema de la petición:


1 0
'M' 'D' 'I' 'R' 0
'D' 'I' 'R' '1' 0
....
1 0
....
1 0

9.3. Programa servidor.

El programa servidor, desarrollado en C, correrá en una máquina Sun con un sistema operativo de arquitectura UNÍS (Solaris). Debe atender las peticiones que reciba por parte de los clientes.

Se ha dicho ya que el servidor sólo atenderá una petición por conexión aceptada. Puede que esto se vea como una desventaja, ya que un cliente, si quisiera descargar dos imágenes distintas, tendría que establecer la conexión con el servidor dos veces, cuando en realidad esto no haría falta. La razón por la que se optó por que el servidor únicamente acepte una petición por conexión, es debido a Java. En unas primeras pruebas se vio que el cliente desarrollado en Java no era capaz realizar varias peticiones correctamente al servidor con una misma conexión. Ahora bien, con exactitud no puedo decir que absolutamente toda la culpa recaiga sobre el lenguaje en si, puesto que ante los primeros errores en la comunicación, opte por la otra opción.

Como ventaja de tener una petición por conexión, tenemos que el servidor esta menos sobrecargado. Si tuviésemos la misma conexión por cliente, y tuviésemos 20 clientes (por poner un ejemplo) conectados simultáneamente, tendríamos un proceso por cada conexión (20 procesos), y no finalizaría ninguno hasta que el programa del cliente finalice y corte la conexión con el servidor. Es decir, tendríamos conexiones con clientes, aunque estos no se estuvieran bajando ninguna imagen o directorio.

El núcleo principal del programa servidor es un bucle infinito que esta continuamente aceptando conexiones nuevas. Para cada conexión nueva que admita, crea un proceso hijo, el cual se encargará de atender las peticiones del cliente asociado a la conexión, mientras el padre sigue aceptando nuevas conexiones.

En realidad, no hubiese hecho falta utilizar varios hilos en nuestro programa servidor, ya que solo es necesario una petición por conexión. Pero debido a que atender determinadas peticiones puede conllevar un tiempo de ejecución que, aunque no elevado, sí considerable, se ha optado por esta opción. Además, garantizamos la máxima velocidad de atención al cliente.

El programa servidor utiliza funciones exclusivas de la API de UNIX, por lo que, aunque este escrito en un C estándar, si se desea pasarlo a otro sistema operativo, habrá que tener conocimiento acerca de la funcionalidad de cada función que se emplea en el programa para luego poder encontrar su equivalente en el sistema operativo al que se transporte. De igual modo, no es una tarea en exceso compleja.

9.4. Programa cliente.

9.4.1. Funcionamiento. Utilización.

El programa cliente ha sido implementado con el lenguaje Java, compatible con versiones que va desde la 1.1.5 hasta las más modernas. Para la interfaz gráfica se han utilizado exclusivamente los paquetes AWT estándar, y cualquier nuevo componente gráfico que se pueda observar en la interfaz, ha sido diseñado para este proyecto. No se ha utilizado ningún componente de terceras personas, no por desconfianza, sino porque eran componentes grandes, con muchas funcionalidades, muchas de las cuales no se iban a utilizar en el programa, y que ocupaban igualmente espacio. Recordemos que, para poder ejecutar un programa en Java, en este caso un applet, es necesario descargar el código a través de Internet, por lo tanto, si tenemos un componente encapsulado en una clase, aunque sólo utilicemos una única parte, se descargara de igual modo toda la clase entera.

Además, esta el tema de las licencias, ya que muchos de los componentes disponibles en Internet son shareware, o tienen licencias extrañas. Así que para evitar cualquier posible problema, todos los componentes existentes en el proyecto, no nativos de Java, son totalmente exclusivos y originales.

El programa cliente esta diseñado para ser incluido en una pagina Web, mediante la cual acceder a dicho programa. El programa va encapsulado en un applet. Es la única forma de ofrecer un programa gráfico en Java (sin poseer el JDK, claro está).

El aspecto de la interfaz de usuario es el siguiente:

En ella se pueden observar una serie de botones en la parte superior, debajo de los cuales existe un cuadro para entrada. Abajo del todo tenemos una pequeña barra de estado que mostrará al usuario información del estado del programa y de la transmisión, así como de las imágenes cargadas.

En el medio esta el área de visualización, donde veremos la imagen que se desee cargar o bien el mosaico, o conjunto de imágenes, del directorio a cargar.

El botón de 'Cargar' mandará la cadena que halla en ese momento en la entrada de texto al servidor, y espera a la confirmación de la petición. Si ha sido confirmada, en función de lo que sea (o imagen o directorio), mostrará progresivamente una imagen, o las imágenes del directorio, en forma de mosaico.

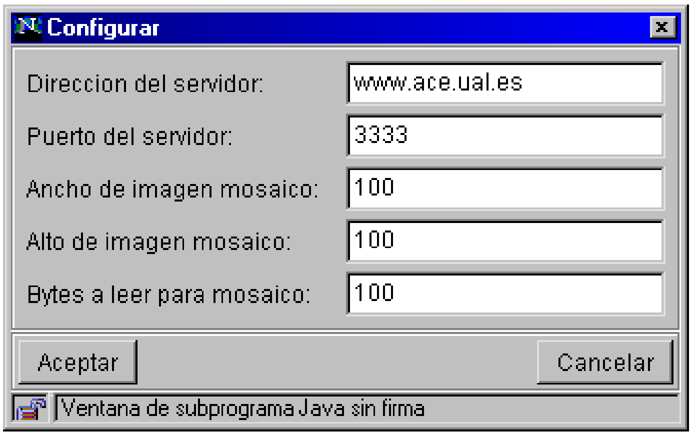
Veamos un ejemplo de cómo quedaría el programa cargando una imagen:

En cualquier momento de la visualización-transmisión de la imagen, podemos parar la misma con el botón de 'Parar'. Dicho botón sirve tanto para parar la recepción de una imagen como de un directorio.

Cuando cargamos un directorio, el aspecto que presenta es el siguiente:

En el caso de un directorio, se nos mostrará un mosaico con todas las imágenes que ha enviado el servidor. Al pasar el ratón por encima de una imagen determinada, se nos presentará en la barra de estado el nombre de la misma (junto con su ruta relativa). Si pinchamos en una de ellas, automáticamente se introducirá su nombre en la entrada de texto y procederá a cargar dicha imagen.

El ancho y el alto de las imágenes mosaico, así como el porcentaje a recibir de cada una de ellas, es posible configurarlo con el botón ‘Configurar’, mostrándose el siguiente cuadro de diálogo:



En este panel podemos especificar la dirección del servidor, el puerto donde estará escuchando el programa servidor, el alto de la imagen mosaico, el ancho de la imagen mosaico y el porcentaje de bytes, de 0 a 100, con posibilidad de dos cifras decimales, a recibir por cada imagen mosaico.

Si se quieren confirmar los datos se pulsara en ‘Aceptar’. En caso contrario, se pulsara en ‘Cancelar’ haciendo que las modificaciones hechas no sean validas.

El botón ‘Directorio’ nos mostrara la siguiente ventana:

En ella podremos ver los directorios que cuelgan del directorio en el que se encuentra el servidor. Si pinchamos sobre cualquiera de ellos, su nombre se introducirá automáticamente en la entrada de texto, y procediéndose seguidamente a la carga de dicho directorio.

El boton de 'Ayuda' mostrará una página web con información acerca del visualizador, así como con ayuda acerca de su manejo.

Para incluir el applet en una página Web, se empleara la etiqueta <APPLET>. Un ejemplo sería:

```
<APPLET CODE="Proyecto.class" WIDTH="0" HEIGHT="0">
<PARAM NAME="Servidor" VALUE="localhost">
<PARAM NAME="Puerto" VALUE="3333">
<PARAM NAME="AnchoMosaico" VALUE="100">
<PARAM NAME="AltoMosaico" VALUE="100">
<PARAM NAME="BytesPorMosaico" VALUE="100">
</APPLET>
```

Como se puede apreciar, al applet es necesario pasarle una serie de parámetros, cuya función puede verse en la siguiente tabla:

Parámetro	Descripción
Servidor	Indica la dirección del servidor.
Puerto	Indica el número de puerto en el que está escuchando el servidor.
AnchoMosaico	Alto de las imágenes de los mosaicos.
AltoMosaico	Ancho de las imágenes de los mosaicos.
BytesPorMosaico	Porcentaje (* 100) de bytes a leer por cada imagen del mosaico.

El applet no tiene dimensión, por lo que es aconsejable poner WIDTH y HEIGHT a 0. El applet muestra una ventana para la ejecución, no utiliza el espacio gráfico reservado para él en la página web.

9.4.2. Detalles técnicos del programa.

El programa consta de un total de 19 archivos de código fuente Java. Muchos de ellos son para definir nuevos controles gráficos, como CustomPanel, ImageButton, BorderPanel o EtchedPanel. Los ficheros que contienen la gran mayoría del código son ImageLoader, MosaicLoader, SPITHDecoder, SRutines2d y Proyecto (con su extensión .java).

El fichero ImageLoader.java contiene la clase encargada de recibir del servidor la imagen y visualizarla progresivamente en la pantalla. Para ello, crea dos hilos, uno encargado de, mediante la clase SPITHDecoder, ir descodificando la imagen que va llegando poco a poco, y otro que se encarga de ir actualizando la pantalla, empleando los datos que va produciendo el hilo anterior y las rutinas de la clase SRutines2d.

El principal problema con que nos encontramos es que en algunas máquinas virtuales, la ejecución de ambos hilos no es simultanea, ni tan sólo en apariencia. Nosotros queremos que muestre un frame cuanto antes mejor, para lo cual, le daremos al hilo que actualiza en pantalla mas prioridad que al otro.

Otro problema, este ya no debido a la maquina virtual, sino de la propia definición de Java, es la que nos podemos encontrar con el método repaint() de un componente. Es usual caer en la trampa de creer que este método "repinta" de inmediato el componente sobre el cual se aplica. Nada mas lejos de la realidad, lo que hace este método, que por cierto es muy abundante en numerosas publicaciones, es mandar un evento al componente indicado de repintar. Pero el tratamiento de este evento por parte del componente no tiene por que ser inmediato, sino que dependerá del numero de hilos que haya en ese momento ejecutándose y del numero de eventos que haya en la cola del componente. Por lo cual, a menos que la maquina virtual sea muy eficiente, nos podemos encontrar con que el frame no aparece en pantalla hasta que se baja toda la imagen y termina el otro hilo su ejecución. Para solventar esto, se fuerza el pintado en pantalla durante la visualización progresiva, mediante el método getGraphics() y llamando directamente a update() del componente que queramos.

La clase MosaicLoader es una variante de ImageLoader, pero no tiene visualización progresiva de las imágenes, ya que en la mayoría de las ocasiones serán de dimensiones reducidas y no será necesario emplear el mecanismo complejo de ImageLoader.

Por ultimo, comentar que la comunicación cliente-servidor se hace siempre byte a byte, nunca se transmite una cantidad mayor de un golpe. La razón, otra vez Java, que tras unas pruebas iniciales resulto prácticamente incompatible, ya que había momentos en los que perdía información.

10. Posibles aplicaciones y desarrollos futuros.

Como se habrá podido observar, el proyecto en cuestión resuelve perfectamente el problema para el cual se diseño, es capaz de mostrar progresivamente imágenes a una velocidad considerablemente alta, con independencia del ancho de banda empleado y de la arquitectura empleada por el cliente para dicho proceso.

Sin embargo, no se aplican algoritmos propios de lo que seria un tratamiento astronómico (calculo de distancias entre estrellas, determinación de la posición real de un punto, etc.). Aquí se da una base genérica para lo que sería una aplicación especifica de tele-adquisición y posterior tratamiento de imágenes.

Otro tema importante a resaltar es que, de momento, las imágenes a transmitir, debido a las propiedades del algoritmo de compresión, deben ser cuadradas. De todas formas, para el visualizador esto es transparente, ya que le da lo mismo que la imagen sea cuadrada o no. Sin embargo, el compresor solo trabaja con imágenes cuadradas. En cualquier caso, modificar el compresor para que trabaje con imágenes de cualquier dimensión, tan solo requiere ciertas modificaciones en el mismo no excesivamente complejas.

Aunque AWT es idóneo para el cliente, no nos asegura una uniformidad en la presentación de nuestra interfaz, ya que puede variar, aunque ligeramente tan solo, de un entorno gráfico a otro. Seria por tanto mas adecuado desarrollarlo con las nuevas clases del paquete Swing de Java 2, aunque con eso limitáramos el abanico de posibles arquitecturas clientes. Aunque esto ultimo tampoco es en exceso preocupante debido al crecimiento y difusión exponencial de Java, y en particular de la nueva versión Java 2.

11. Bibliografía y fuentes de información.

- Javier Garcia de Jalon, Jose Ignacio Rodriguez, *“Aprenda Java como si estuviera en primero”*. Escuela superior de Ingenieros Industriales. Universidad de Navarra.
- Steven Holzner, *“La biblia de Java 2”*. Anaya Multimedia.
- Elliotte Rusty Harold, *“Los secretos de Java a tu alcance”.* Anaya Multimedia.
- Amir Said, William A. Pearlman, *“A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees”*. IEEE Transactions on Circuits and Systems for Video Technology. Vol. 6, Junio 1996.
- Abrahams y Larson, *“Unix para impacientes”*.
- Stevens, *“Advanced Programming inthe UNIX environment”*.
- http://marathon.csee.usf.edu/~mpowell/javaip.html
- http://www.markschulze.net/java/index.html
- http://www.javasoft.com/docs/books/tutorial/2d/index.html
- http://ipl.rpi.edu/SPIHT
- http://www.digitalcats.com