# INGENIERÍA DE TELECOMUNICACIÓN

**Departamento de Ingeniería de Sistemas y Automática**

**Escuela Superior de Ingenieros**

**Universidad de Sevilla**

# PROYECTO FIN DE CARRERA

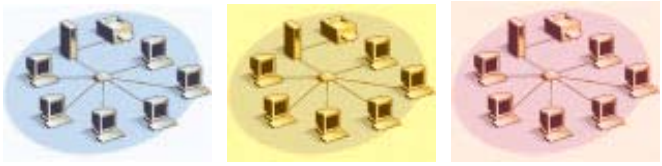
## "TRANSMISIÓN DE IMÁGENES A TRAVÉS DE RED ETHERNET SIGUIENDO EL MODELO CLIENTE SERVIDOR"

**AUTOR:** Fernando Gabriel Cornello Sánchez

**TUTORA:** Begoña C. Arrue Ullés

**MAYO 2003**

# ÍNDICE



# CAPÍTULO 1

# INTRODUCCIÓN

# 1 INTRODUCCIÓN

## 1.1 OBJETIVOS DEL CAPÍTULO

El objetivo de este capítulo de introducción es poner de manifiesto el propósito fundamental que se pretende conseguir mediante la realización de este proyecto fin de carrera. Una vez que se tenga presente lo que se pretende conseguir, se introduce al lector durante el desarrollo de la explicación en la situación actual del sistema donde se va a implantar la aplicación y se detallan las mejoras conseguidas con la introducción del software desarrollado en este trabajo. Al final de este capítulo, se muestra la estructura de desarrollo que se va a seguir a lo largo de esta memoria, de forma que en todo momento, se tenga conciencia de qué se pretende explicar y por qué hemos considerado necesario hacerlo.

## 1.2 OBJETIVOS DE ESTE PROYECTO FIN DE CARRERA

El presente proyecto fin de carrera, realizado en el Departamento de Ingeniería de Sistemas y Automática, pretende desarrollar un sistema de transmisión de imágenes a través de red Ethernet siguiendo el modelo Cliente-Servidor. Para llevar a cabo este cometido se han diseñado e implementado las correspondientes aplicaciones Cliente y Servidor para los casos de funcionamiento siguientes:

- Transmisión de imágenes sin compresión.
- Transmisión de imágenes con compresión.
- Manejo de la digitalizadora Matrox Meteor II Multichannel (sin compresión).
- Manejo de la digitalizadora PC-COMP de la compañía Imaging Technology (con y sin compresión).

El proyecto se ha realizado teniendo presentes dos fines fundamentales:

- Poner a disposición de todo aquel que lo necesite, un conjunto de aplicaciones que permita la transmisión de forma rápida y eficaz de imágenes a través de un entorno de red de área local (LAN). Se dispondrá de un *Servidor de Imágenes*, que es el programa que soporta la mayor parte del proceso y de la aplicación *Cliente*, que dispone de un interfaz amigable para que incluso una persona no necesariamente experta pueda emplearlo.

- Satisfacer la necesidad concreta de incluir, en el proyecto de detección de humo que también se está desarrollando en este departamento, la posibilidad de trabajar con la tarjeta que digitaliza las imágenes que captura la cámara de forma independiente con respecto a su localización física. Es decir, como se describe en detalle en el capítulo 2 de este proyecto, se trata de evitar la exigencia que tienen las aplicaciones que necesitan de la digitalizadora, de funcionar en el mismo equipo donde esta se encuentra instalada. Por este motivo, se han implementado los programas que componen este trabajo de tal forma que su integración en otros estudios de mayor índole fuera lo más sencilla posible.

De hecho, este proyecto no sólo trata del desarrollo de las aplicaciones, sino que se ha realizado también la integración de estas en el proyecto de detección de humo anteriormente mencionado.

Se realiza también al final del proyecto un análisis final del estado de la red que se haya instalada en el departamento en virtud de los resultados obtenidos en los ensayos.

## 1.3 ESTRUCTURA DEL DOCUMENTO

Este documento está compuesto por nueve capítulos, los cuales, junto con la sección de referencias bibliográficas constituyen la memoria de este Proyecto Final de Carrera. La disposición de estos capítulos no ha sido en ningún momento resuelta de forma aleatoria, sino que se ha realizado siguiendo el orden que se ha estimado más oportuno para facilitar su comprensión y posterior utilización en caso se que se pretendan realizar futuras consultas.

### 1.3.1 CAPÍTULO 1

Esta memoria comienza con este capítulo en el que se introducen de forma clara los objetivos del trabajo desarrollado, incluyendo la estructuración del presente documento.

### 1.3.2 CAPÍTULO 2

En el segundo capítulo se realiza una descripción del entorno de trabajo donde se va a llevar a cabo el diseño e implementación de las aplicaciones, haciendo de nuevo hincapié en las necesidades específicas que se pretenden satisfacer y las soluciones

adoptadas para realizar este cometido. Este capítulo es de vital importancia en esta memoria ya que las necesidades que se han de satisfacer quedan perfectamente descritas por la disposición actual de los distintos elementos que intervienen en el entorno donde van a funcionar las aplicaciones desarrolladas.

### 1.3.3 CAPÍTULOS 3 Y 4

Puesto que uno de los objetivos de este proyecto es ponerlo a disposición de personas no expertas en problemas y situaciones de ingeniería, se ha estimado oportuno incluir los capítulos tres y cuatro, que hacen respectivamente, una descripción de los conceptos fundamentales relativos a las redes de área local y al protocolo de transmisión TCP/IP. Como veremos más adelante, el resultado final de las aplicaciones presenta un intefaz sencillo a la vez que práctico, que permite desde la aplicación Cliente hacer un uso adecuado del Servidor de imágenes. El fin de estos capítulos es que se puedan seguir las distintas explicaciones que se desarrollan lo largo de esta memoria.

### 1.3.4 CAPÍTULO 5

Como parte de las aplicaciones que se han implementado emplean compresión de imágenes, se han incluido en el capítulo cinco de la memoria, las conclusiones a las que se ha llegado tras el estudio de los distintos algoritmos de compresión que se han barajado durante la fase de desarrollo explicando en todo momento además de su funcionamiento, los motivos que nos han llevado a decidir por unos sobre otros. Se introduce además al lector menos experimentado en los fundamentos de la compresión con pérdidas y sin pérdidas.

### 1.3.5 CAPÍTULO 6

Otra de las partes importantes de este proyecto fin de carrera ha sido el empleo de las tarjetas digitalizadoras de las compañías Matrox e Imaging Technology. Éstas serán las encargadas de digitalizar la información que capturan las cámaras y que posteriormente se trasmite. Se detallan, por tanto, en el capítulo seis las características de funcionamiento de estas dos tarjetas empleadas, así como, a modo de breve manual de usuario, aquella parte del software que proporciona el fabricante y que consideramos más útil a la hora de desarrollar nuevas aplicaciones.

### 1.3.6 CAPÍTULO 7

En el capítulo siete de la memoria se describe la aplicación Cliente, explicando de forma detallada al lector, las soluciones adoptadas a la hora de afrontar las distintas necesidades que debe satisfacer el trabajo. Se hace además una breve explicación de las características fundamentales de la programación en Visual C++, con el objeto de que el lector no experimentado en la programación pueda seguir de manera más o menos interesada el desarrollo de la explicación. Finalmente, en este capítulo se detalla el funcionamiento del interfaz gráfico de la aplicación Cliente, que engloba la práctica totalidad de la funcionalidad de la que dispone la citada aplicación.

### 1.3.7 CAPÍTULO 8

El capítulo ocho es el análogo del anterior pero para el caso de la aplicación Servidor. Además para facilitar el seguimiento de la explicación se ha conservado la misma estructura de desarrollo que en el caso del capítulo correspondiente a la aplicación Cliente.

### 1.3.8 CAPÍTULO 9

Finalmente, en el capítulo nueve se describen las conclusiones a las que se ha llegado tras la realización de las pruebas de las aplicaciones desarrolladas, realizando dicho estidio para un conjunto de imágenes de muy distintas características y funcionando en el modo con compresión y sin compresión. Se aclaran finalmente las posibles líneas de investigación que pudieran surgir tomando este proyecto final de carrera como base para la citada línea de desarrollo.

# CAPÍTULO 2

# ENTORNO DE TRABAJO

# 2 ENTORNO DE TRABAJO

## 2.1 OBJETIVOS DEL CAPÍTULO

El propósito de este capítulo de la memoria es, una vez introducidos los objetivos del proyecto fin de carrera, describir las características de los sistemas empleados para llevarlo a cabo. Detallaremos por tanto el material y equipos de los que se ha dispuesto a la hora de afrontar las líneas de desarrollo del proyecto y las necesidades surgidas en virtud a los distintos sistemas en los que hemos trabajado.

De forma paralela, se describirá el entorno en el cual se van a integrar los trabajos desarrollados y para los que se ha realizado este proyecto.

Finalmente y una vez que el lector tenga una idea adecuada de los distintos subsistemas, se procederá a explicar la solución adoptada para efectuar la generación, implementación y prueba de los programas que componen este proyecto fin de carrera.

## 2.2 ESTACIÓN DE TRABAJO

En primer lugar vamos a detallar el lugar y equipos donde de ha llevado a cabo el desarrollo de los trabajos.

La estación de trabajo está constituida por un PC con las siguientes características:

- Microprocesador Intel Pentium II a 300 Mhz.
- 296 Mbytes de Memoria RAM.
- Monitor DXC de 17’
- Dos unidades de disco duro.
- CD-ROM.
- Tarjeta Ethernet a 10 Mbps para la conexión de Red.
- Tarjeta Digitalizadora PC-COMP de la compañía Imaging Technology. Las características concretas de este dispositivo se describen exhaustivamente en el capítulo 6 de este documento.

- Sistema operativo instalado: Windows 2000 NT.
- Programa de desarrollo de las aplicaciones: Microsoft Visual C++ 5.0.

Estas características se han de tener presentes a la hora de interpretar los resultados de las pruebas que se realizan para probar la efectividad del trabajo desarrollado y que se detallan en el capítulo 9 de la memoria.

Se dispone además en el puesto de trabajo de una cámara visual en color de la compañía JAI que es la que se va a utilizar para realizar los distintos ensayos antes de la implantación del proyecto en el sistema del posicionador. Al igual que la tarjeta digitalizadora, describiremos con más detenimiento las características de esta cámara en el citado capítulo 6.

Todo el conjunto se encuentra en el Laboratorio de Proyectos, situado en la segunda planta del edificio de talleres y laboratorios del Departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingenieros de Sevilla.

## 2.3 DESCRIPCIÓN DEL SISTEMA POSICIONADOR

El sistema posicionador va a ser uno de los destinos finales del trabajo desarrollado en este proyecto fin de carrera. Es decir, aunque los programas se han implementado en la estación de trabajo descrita en el apartado anterior, hemos tenido que hacer una adaptación final de los códigos para poder integrarlos en el sistema donde se requería. Esta portabilidad de los programas, en lo referente a la interacción con el hardware, es uno de los requisitos que hemos tenido presente en todo momento a la hora de realizar el proyecto.

El sistema completo está formado por el posicionador, una cámara visual, otra de infrarrojos (ambas montadas sobre dicho posicionador) y dos PC's. Tanto las cámaras como el posicionador se encuentran en la azotea del edificio de los laboratorios de la Escuela de Ingenieros de Sevilla. Todo el control de los equipos se realiza a través de los ordenadores PC's. Uno de ellos se encuentra en la última planta del citado edificio y el otro en la planta baja.

El motivo por el que se emplean dos PC's en el montaje (que no es en ningún caso objetivo del presente trabajo), es para simplificar las tareas de cableado, ya que

este sería muy complejo si se tratara de controlar el posicionador desde el ordenador de la planta baja.

El sistema posicionador presenta dos grados de libertad (elevación y azimut) y es de la empresa Jenny Electronics AG, modelo JENYTEC ST-40V5, compuesto por dos motores de continua con funcionamiento por pasos. Para la gestión de estos motores se emplean conexiones RS-485. Por tanto se necesitarán convertidores RS-232 a RS-485 para controlarlos a través del PC.

La cámara visual pertenece a la compañía JAI, concretamente el modelo MCL-1500 DSP Color con Zoom x10 gestionada a través de Interfaz RS-232.

En cuanto a la cámara de infrarrojos su gestión se realiza a través de las tarjetas PCL-812PG y PCL-720 las cuales se encuentran instaladas en el ordenador situado en la planta alta.

**Figura 2.1. Sistema Posicionador.**

El ordenador que hay en la planta baja tiene las siguientes características:

- Microprocesador Intel Pentium III 700 Mhz.

- 128 Mbytes de Memoria Ram.
- Monitor de 15'.
- Dos unidades de disco duro.
- CD-ROM.
- Tarjeta Ethernet a 10 Mbps para la conexión de Red.
- Tarjeta Digitalizadora MATROX METEOR II MULTICHANNEL de la compañía MATROX. Las características concretas de este dispositivo se describen exhaustivamente en el capítulo 6 de este documento.
- Sistema operativo instalado: Windows NT.
- Programa de desarrollo de las aplicaciones: Microsoft Visual C++ 5.0.

En cuanto al ordenador que encontramos en la planta superior del edificio diremos que este se encuentra instalado en un armario en el que además del PC encontramos la fuente de alimentación que alimenta al posicionador y a las cámaras junto con los convertidores RS-232 a RS-485.

Las características de este equipo son:

- Microprocesador Intel.
- Memoria Ram.
- Monitor de 11'.
- Unidad de disco duro.
- Tarjeta Ethernet a 10 Mbps para la conexión de Red.
- Las tarjetas PCL-812PG y PCL 720 para la gestión de la cámara IR.
- Sistema operativo instalado: Windows NT.

Una vez explicados los equipos presentes en el sistema y que han intervenido en la realización del trabajo, vamos a describir la forma en que están conectados para así poder hacernos una idea de la funcionalidad de cada uno de estos componentes.

En primer lugar tenemos el ordenador de arriba, que como hemos dicho controla todo el sistema posicionador según las instrucciones que se le envían desde el PC de la planta baja a través de la red Ethernet que se encuentra instalada en el laboratorio. Las cámaras transmiten la información que capturan a través de los correspondientes cables coaxiales que las conectan directamente con la tarjeta Matrox del ordenador de la planta baja, y es esta tarjeta la que se encarga de procesar, representar y almacenar la información que proviene de estas cámaras.

## 2.4 RED DE ÁREA LOCAL

La red de área local que se encuentra instalada en el edificio de los laboratorios es un elemento de particular importancia en este proyecto fin de carrera. Nuestro trabajo se ha desarrollado para ser implementado en esta red y es por ello por lo que es necesario conocer sus características, con el fin de ajustar de la forma más adecuada posible nuestro diseño.

Hemos de reseñar en este punto, que el montaje u optimización de la red no ha sido en ningún momento objeto de este proyecto fin de carrera. La red se encontraba instalada antes del comienzo de este trabajo. Lo que sí se ha tenido en cuenta a lo largo de todo el proceso de desarrollo es procurar un diseño que aproveche al máximo las características de la red así como un esquema económico en cuanto a recursos consumidos, con el fin de agilizar las tareas y no aumentar demasiado el tráfico que circula por la red.

Esta red que hay instalada en el edificio de laboratorios es una LAN tipo Ethernet a 10 Mbps cuya configuración espacial es la conocida como topología en bus. El cableado empleado para la conexión de los distintos equipos es el par trenzado sin apantallar o UTP. Los conectores son los típicos RJ-45 que se usan con este tipo de cableado. Esta red cumple con la norma 802.3 correspondiente como hemos dicho anteriormente al protocolo Ethernet.

Para más información con respecto a las redes de área local remitimos al lector al capítulo 3 del documento. La comprensión de los términos aquí empleados será de vital importancia a la hora de interpretar los resultados y conclusiones a las que se llega más adelante en este trabajo.

## 2.5 ESTUDIO DE LAS NECESIDADES

Una vez descrita la infraestructura existente, para realizar la tarea para la que ha sido construida, salta a la vista que existen múltiples posibilidades de mejora. La realización de uno de estos avances es lo que nos ha llevado a trabajar en este proyecto fin de carrera.

Es evidente que la disposición actual del sistema es completamente ineficiente a la hora de implantarlo en entornos donde no se puede tener cerca físicamente los dos ordenadores y el posicionador. Es decir, se necesitan tres cables, dos de ellos cables coaxiales (cuyo coste es bastante elevado) además del cable de red para conectar el sistema completo. Si tenemos que instalar las cámaras en un lugar alejado del ordenador donde se va a realizar el procesado de las imágenes tendremos que comprar cables de gran longitud y hacer la instalación de dicho cableado. Asimismo, si en un futuro queremos añadir más cámaras al sistema hemos de añadir más cableado y reinstalarlo, con el correspondiente trabajo que ello conllevaría.

Igualmente tendremos la obligación de instalar y ejecutar la aplicación concreta que utilice las imágenes provenientes de las cámaras, en el mismo ordenador donde se encuentre instalada la tarjeta digitalizadora ya que es esta es la encargada de capturar y digitalizar la información, almacenándola en el equipo donde se encuentra montada. O bien hemos de trasladar la información mediante algún tipo de soporte informático con la correspondiente molestia que ello conlleva, sobre todo porque las imágenes suelen ser archivos de gran tamaño.

Aquí es donde entra en juego la aplicación Servidor. Nuestro programa se ha de implantar en el equipo que posee la tarjeta digitalizadora, para así poder suministrar a la aplicación Cliente, que puede estar funcionado en cualquier otro equipo de la red de área local, las imágenes que en ese momento esté grabando la cámara, imágenes individuales o una secuencia de imágenes previamente guardadas en disco o almacenadas por la tarjeta. Todo esto según las necesidades del cliente en cada momento. Por tanto, es importante destacar que la aplicación Cliente es completamente independiente de la máquina de la red donde se implemente (ya no tiene la obligación de ejecutarse donde está instalada la tarjeta digitalizadora). Y más aún, el programa servidor también lo es, siempre y cuando se instale en un ordenador donde exista una tarjeta digitalizadora de las compañías anteriormente

citadas. Podemos tener incluso varios programas Servidores ejecutándose en maquinas distintas y poder acceder desde un cliente a cualquiera de estos Servidores únicamente conociendo la dirección IP del equipo donde se esté ejecutando y número del puerto a través del cual queremos que se comuniquen.

Por otra parte, en vez de tener que utilizar tres cables para que el sistema funcione (los dos de las cámaras y otro de red) solamente haría falta instalar uno (el de red) previa instalación de la tarjeta digitalizadora en el ordenador de la planta alta. Además de la portabilidad de que ahora dispondría el sistema, ya que no dependería la longitud de los cables, tal y como se ha detallado anteriormente, solamente habría de disponer de un punto de acceso a la red interna.

Por tanto y una vez explicado lo anterior, podemos afirmar que con los programas desarrollados se produce una sustancial mejora en la infraestructura de las instalaciones, además de abrir un amplio abanico de posibilidades a la hora de generar aplicaciones o trabajos basados en el sistema posicionador y todo lo que ello conlleva.

Tampoco podemos olvidar que ambos programas son una importante herramienta software por sí solos. Estas aplicaciones constituyen en conjunto un método eficaz de acceso a una base de datos de imágenes que se encuentren en otro equipo dentro de nuestra red, ya que el programa Servidor no necesita que el equipo donde reside tenga instalada una tarjeta digitalizadora. Simplemente lo que ocurriría es que las funciones referentes a captura de imágenes desde la cámara no tendrían ninguna utilidad y las debemos omitir, pero en cambio la transferencia desde archivos almacenados en disco y la transmisión de secuencia de imágenes funcionarían correctamente. Es por ello por lo que para estas aplicaciones, se ha desarrollado en este proyecto fin de carrera un interfaz gráfico claro y conciso, para que cualquier usuario (no necesariamente experto), pueda trabajar de forma cómoda con los programas. Este interfaz se describe detalladamente en capítulos posteriores de este documento.

Cabe citar en este apartado que el código de la aplicación está diseñado en Visual C++ y esto implica que se trata de un programa fuertemente estructurado (remitimos en este punto al lector al apartado 2.6 para ver las características que distinguen a este lenguaje de programación). Al ser así podemos extraer del código las partes del

mismo que nos interesen. Por citar un ejemplo, la clase que se encarga de la interacción con la tarjeta digitalizadora podemos trasladarla a otras aplicaciones que necesiten hacer uso de ésta. Esto multiplica la utilidad de las aplicaciones implementadas.

Está claro que existen en el mercado herramientas software que se encargan de transmitir imágenes a través de una red de área local. Pero hemos de tener en cuenta que en este caso hemos de satisfacer unas necesidades muy concretas, debiéndose así generar un software muy personalizado. Estas necesidades podemos resumirlas en los siguientes puntos:

- En primer lugar, el programa Cliente ha de ser perfectamente portátil. Es decir, se ha de hacer lo más independiente posible de la aplicación en concreto con el fin de que todas las aplicaciones que se desarrollen para trabajar con las cámaras instaladas puedan de una manera sencilla y eficaz incorporar el programa o el código de éste.

- Por otra parte tenemos que los requerimientos hardware también muy específicos. El programa Servidor debía ser implementado para trabajar con las tarjetas digitalizadoras que se encontraban ya compradas e instaladas en el laboratorio. Recordemos que estas tarjetas son la de la compañía Matrox y la de la compañía Imaging Technology. Es posible encontrar programas comerciales que contemplan este hardware pero además incorporan software para trabajar con otras muchas tarjetas. Es decir, que tendríamos un programa mucho más complejo de lo estrictamente necesario. También se encuentran disponibles en el mercado aplicaciones que trabajan con archivos. Pero para la necesidad concreta de transmitir la imagen que ese momento está grabando la cámara no es necesario guardar en disco dicha información. Es más, resulta en ocasiones menos eficiente guardar en disco antes de enviar porque las operaciones de lectura y escritura son generalmente lentas y no sería por tanto la alternativa más eficiente. Si exigiéramos programas para trabajar con estas digitalizadoras concretas o bien que transmitan imágenes sin partir de información almacenada en disco estaríamos exigiendo a la empresa vendedora del software, una aplicación a medida, con el considerable incremento de precio que ello implicaría.

- Tendremos también como requisito indispensable para trabajar con la aplicación de detección de humo que el software sea capaz de transmitir secuencias de imágenes almacenadas en disco teniendo el cliente únicamente que especificar la ruta y nombre del archivo de la primera de las imágenes que componen esta secuencia.

- Con el fin de poder trabajar con la red de área local instalada en el laboratorio (Véase apartado 2.4) hemos de implementar una aplicación que utilice esta red de manera sencilla y eficaz. Por ello, tal y como se explicará más adelante en este capítulo, se trabajará con los sockets de Windows. Además, para no sobrecargar en exceso la red en los casos de tráfico masivo, se han introducido en la aplicación la posibilidad de comprimir las imágenes para la transmisión. Dada la importante función que ha de llevar a cabo la aplicación (detección de humo), hemos considerado que cualquier pérdida de información de la imagen podría ser perjudicial, de ahí que el método implementado sea un método de compresión sin pérdidas.

- Como toda aplicación software que se precie, hemos creído conveniente el incluir como requisitos de la aplicación Cliente, la posibilidad de guardar en disco la imagen que se está visualizando en este momento y también la posibilidad de rescatar de disco una imagen previamente guardada.

- Tenemos también que considerar como motivación para este trabajo que los equipos informáticos que existen en el laboratorio no son de última generación. Ello supone que si tenemos programas grandes que realizan tareas complejas los sistemas pueden saturar dejando incluso de funcionar. Si a estos programas complejos le hemos de añadir además una aplicación cliente que sea también extensa y que realice muchas funciones que no utilizamos, la situación sería incluso peor. Es por ello por lo que al desarrollar este proyecto se ha estudiado e implementado solamente aquella funcionalidad estrictamente necesaria para cubrir completamente las necesidades requeridas. Además al disponer del código fuente de los programas podemos estudiar su funcionamiento y a su vez podemos tomar de ellos solamente la parte que nos interese con el fin de no sobrecargar las aplicaciones con código inservible. La mayoría de las empresas dedicadas al software no ponen a disposición de los usuarios o clientes el código fuente de sus aplicaciones y las que lo hacen lo venden a precios muy elevados.

- No podemos tampoco obviar en ningún momento que este trabajo es un proyecto fin de carrera realizado por un alumno que ha de culminar su formación universitaria. Esto supone que el proyecto ha de tener un fin didáctico además de práctico. Este aspecto ha resultado del todo satisfactorio en lo personal, ya que durante el estudio inicial, el desarrollo y finalmente la implantación del trabajo se han cumplido con creces las expectativas generadas en cuanto a la formación personal que debería proporcionar un proyecto fin de carrera.

## 2.6 SOLUCIÓN ADOPTADA

En este punto y una vez descritas las necesidades y motivaciones que nos llevan a desarrollar este proyecto, junto con el entorno de trabajo donde se va llevar a cabo, procedemos a explicar la solución que hemos adoptado. Se van a analizar punto por punto las decisiones que se han tomado, justificándolas en todo momento.

### 2.6.1 EL LENGUAJE DE PROGRAMACIÓN

Cuando desarrollamos un producto software, nuestro deseo es que este sea rápido, fiable, fácil de usar, legible, modular, estructurado y así sucesivamente. Pero estas características describen dos tipos de cualidades diferentes:

- Por una parte distinguimos los llamados *factores de calidad externos,* que son aquellos cuya presencia o ausencia puede ser detectada por los usuarios. Estos factores pueden ser por ejemplo la velocidad y la facilidad de uso entre otros.

- Por otra parte existen otras cualidades aplicables a un producto de software que son perceptibles sólo por profesionales o entendidos en informática que tienen acceso al código fuente. Estos son los llamados *factores de calidad internos* y engloban entre otros a la modularidad o legibilidad.

Debemos tener presente siempre a lo largo de este documento que las técnicas para obtener una buena calidad interna no son un fin en sí mismas, sino un medio para alcanzar las cualidades externas de las aplicaciones que se desarrollan.

Vamos por tanto a centrarnos en las características externas del software, y para ello vamos a definir los siguientes términos que se emplearán frecuentemente lo largo de esta memoria:

- *Corrección:* Corrección es la capacidad de las aplicaciones software para realizar con exactitud la tarea para la que han sido diseñadas. Esta es la cualidad principal. Si un sistema no hace bien lo que se supone que ha de hacer poco importan las demás consideraciones que hagamos sobre él.

- *Robustez:* La robustez es la capacidad de los sistemas software de reaccionar apropiadamente ante condiciones excepcionales. Esta complementa a la corrección. La corrección tiene que ver con el comportamiento de un sistema en los casos previstos para su especificación; la robustez caracteriza todo lo que sucede fuera de tal especificación. Como se refleja en estas palabras, la robustez es por naturaleza una noción más difusa que la corrección. Puesto que tiene que ver aquí con casos no previstos por la especificación, no es posible decir, como con la corrección, que el sistema debería realizar sus tareas en tal caso; donde las tareas son conocidas, el caso excepcional formaría parte de la especificación y regresaríamos al terreno de la corrección. Es habitual que existan casos que la especificación no contemple explícitamente. El papel requisito de la robustez en este caso es asegurar que el sistema no produzca eventos catastróficos; debería producir mensajes de error apropiados, terminar su ejecución limpiamente o entrar en el llamado modo de "degradación elegante".

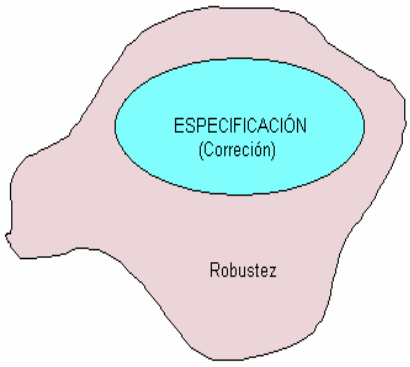



**Figura 2.2. Especificación frente a Robustez.**

- *Extensibilidad*: Es la facilidad de adaptar los productos software a los cambios en la especificación. El problema de extensibilidad es en general un problema de escala. Para programas pequeños realizar cambios no es normalmente una tarea difícil. Pero a medida que la aplicación se hace más grande comienza a ser cada vez más difícil de adaptar. Las técnicas tradicionales de Ingeniería de Software no tienen suficientemente en cuenta el cambio y se basan en una visión ideal del ciclo de vida de la aplicación donde en la etapa inicial de análisis se congelan los requisitos y el resto del proceso se dedica al diseño y la construcción de una solución. Pero ahora en las nuevas técnicas de ingeniería, el cambio es omnipresente en el desarrollo del software: cambios de los requisitos, de nuestra comprensión de los requisitos, de los algoritmos, de la representación de los datos, de las técnicas de implementación... Los dos principios esenciales para mejorar la extensibilidad son la simplicidad en el diseño y la descentralización (módulos lo más independientes posibles). Ofrecer soporte para los cambios es un objetivo básico de la programación orientada a objetos.

- *Reutilización*: Reutilización es la capacidad de los elementos de software de servir para la construcción de muchas aplicaciones diferentes. La necesidad de reutilización surge de la observación de que las aplicaciones siguen patrones similares. Capturando tal patrón, un elemento de software reutilizable se podrá aplicar en muchos desarrollos diferentes. Ello supondrá a la larga un ahorro de trabajo considerable. Además la reutilización tiene influencia sobre los demás aspectos de la calidad ya que al tener que escribir menos código (porque reutilizamos uno anteriormente creado) podremos dedicar mayores esfuerzos a mejorar otros factores como la corrección y la robustez por ejemplo.

- *Compatibilidad*: Es la facilidad de combinar unos elementos software con otros. Es una característica importante dado que las aplicaciones no se desarrollan en el vacío, necesitan interactuar con otras aplicaciones. Pero en general los sistemas tienen dificultades a la hora de esta interacción porque hacen suposiciones contradictorias sobre el resto del mundo. La clave de la compatibilidad recae en la homogeneidad del diseño y en acordar convenciones estándar para la comunicación entre programas.

- *Eficiencia*: La eficiencia es la capacidad de un sistema software para exigir la menor cantidad de recursos hardware, tales como tiempo de procesador, espacio ocupado de memoria interna y externa o ancho de banda utilizado en los dispositivos de comunicación. Existen dos corrientes a la hora de tratar con este problema, los programadores que se obsesionan con el rendimiento y dedican largo tiempo y esfuerzo a su mejora y por otro lado los programadores que priman la corrección del diseño amparándose en que los equipos PC son cada vez más potentes y eficaces. De manera más general podemos decir que la preocupación por la eficiencia debe sopesarse con otros objetivos tales como la extensibilidad o la reutilización. Optimizaciones extremas pueden hacer a la aplicación tan especializada que limite el cambio y la reutilización. Es más, la potencia creciente del hardware de los ordenadores nos permite tener una actitud más relajada con respecto a tratar de ganar hasta el último byte o microsegundo. Pero a pesar de esto, no se debe infravalorar la eficiencia.

- *Portabilidad*: La portabilidad es la facilidad de transferir los productos software a diferentes entornos hardware y software. Muchas de las incompatibilidades existentes entre las plataformas son injustificadas y a veces solamente atienden exclusivamente a criterios de mercado.

- *Facilidad de uso*: Es la facilidad con la cual las personas con diferentes formaciones y aptitudes pueden aprender a usar las aplicaciones desarrolladas y emplearlas para la resolución de problemas. También cubre la facilidad de instalación, de operación y de supervisión. La definición insiste en los diferentes niveles de experiencia de los posibles usuarios. Este requisito plantea uno de los mayores retos de los diseñadores de software preocupados por la facilidad de uso: cómo proporcionar facilidades y explicaciones detalladas a los usuarios menos experimentados sin perjudicar en exceso a los usuarios expertos. Al igual que con otras de las cualidades explicadas detalladas anteriormente, una de las claves de la facilidad de uso es la simplicidad estructural. Un sistema bien diseñado, construido de acuerdo con una estructura clara y bien pensada, tiende a ser más fácil de aprender y usar que uno confuso. De vital importancia a la hora de diseñar una aplicación software fácilmente manejable es el interfaz de usuario. Los buenos diseñadores de interfaces siguen la política de hacer las menos suposiciones posibles sobre los usuarios. Si la aplicación está dirigida a

un área especializada de aplicación, se puede dar por supuesto que los usuarios están familiarizados con sus conceptos básicos pero incluso esta suposición puede ser arriesgada. Según dice Wilfred J. Hansen en su libro 'User Engineering Principles for Interactive Systems': "No suponga que conoce al usuario; realmente no lo conoce".

- *Funcionalidad*: Funcionalidad es el conjunto de posibilidades que proporciona un sistema. Uno de los principales problemas a los que se enfrenta un diseñador es conocer cuánta funcionalidad es suficiente. La presión para ofrecer más facilidades, conocida en el lenguaje de la industria como *featurism*, está constantemente presente. En cambio la aplicación óptima es aquella que realiza la correctamente la tarea para la que ha sido diseñada. Si incluimos a los diseños funcionalidad extra, es decir, funciones que no van a ser necesarias a la hora de utilizar el sistema software, tendremos un descenso en el rendimiento de esta aplicación. Esto se debe a que estaremos consumiendo en primer lugar tiempo de diseño innecesario que podríamos haber empleado en la mejora de otras características. Además, consumiremos también recursos del sistema como memoria o espacio en disco con programas llenos de funcionalidad que no se va a usar durante la utilización normal de la aplicación. Esta funcionalidad "añadida" provocará también el incremento de la complejidad del programa.

- *Oportunidad*: Oportunidad es la capacidad de un sistema software de ser lanzado cuando los usuarios lo desean o antes. Esta es una de las mayores frustraciones de la nueva industria. Un gran producto software que aparece demasiado tarde puede no alcanzar su objetivo. Esto se debe a que pocas industrias evolucionan tan rápidamente como la del software.

- *Verificabilidad*: Es la facilidad para preparar los procedimientos de aceptación, especialmente datos de prueba y procedimientos para detectar fallos y localizar errores durante las fases de validación y operación.

- *Integridad*: Es la capacidad de los sistemas software de proteger sus diversos componentes (programas, datos, etc.) contra modificaciones y accesos no autorizados.

- *Reparabilidad*: Es la capacidad para facilitar la reparación de los defectos.

- *Economía*: Junto con la oportunidad, es la capacidad que un sistema tiene de completarse con el presupuesto asignado o por debajo del mismo.

En lo referente a las características diremos finalmente que existen factores que son claramente opuestos entre sí, con lo que se produce un conflicto a la hora de tratar de mejorar ambos a la vez. Ello nos lleva a tratar de buscar soluciones de compromiso que reconcilien factores en conflicto. Pero si es necesario decidirse por algún factor de calidad la corrección sobresale del resto. No hay justificación para comprometer la corrección en aras de otras cuestiones tales como la eficiencia. Si el software no lleva a cabo su función el resto es inútil.

Una vez que hemos dejado bien claro cuáles son las virtudes que se han de potenciar en una herramienta software, vamos a justificar la elección de un lenguaje de programación orientado a objetos para realizar este proyecto fin de carrera:

- *Corrección y Robustez*: Es bastante difícil producir software sin defectos (bugs) y muy difícil corregir estos defectos una vez que están ahí. Las técnicas para mejorar la corrección y la robustez son similares: Enfoques más sistemáticos para la construcción de software; más especificaciones formales; más comprobaciones integradas a lo largo del proceso de desarrollo del software (no sólo la prueba y depuración después de hecho); mejores mecanismos de lenguajes tales como la comprobación estática de tipos, las aserciones, la gestión automática de memoria y un tratamiento disciplinado de las excepciones permiten a los programadores asegurar los requisitos de corrección y de robustez y posibilitan herramientas para detectar las inconsistencias antes de que de conviertan en defectos. Todas estas herramientas están presentes en la programación orientada a objetos.

- *Extensibilidad y Reutilización*: Como se ha dicho anteriormente el software debe ser fácil de cambiar; las aplicaciones que se produzcan deben ser aplicables de la forma más general posible, debiendo existir un gran inventario de componentes de propósito general que puedan reutilizarse cuando se vaya a desarrollar un nuevo sistema. La programación orientada a objetos permite producir arquitecturas totalmente descentralizadas, cuyos componentes son autocontenidos y se comunican sólo a través de canales restringidos y claramente definidos. Esto es lo que se conoce con el término de *Modularidad*.

- *Compatibilidad*: El método orientado a objetos promueve un estilo común de diseño y módulos e interfaces de sistemas estándares, que ayudan a producir sistemas que pueden interactuar entre sí.

- *Portabilidad*: Con el énfasis en la abstracción y la ocultación de información, la tecnología de objetos estimula a los diseñadores a distinguir entre las propiedades de especificación y de implementación, facilitando los esfuerzos de portabilidad. Las técnicas de polimorfismo y ligadura dinámica harán posible escribir sistemas que se adapten automáticamente a los distintos componentes tanto hardware como software de la máquina.

- *Facilidad de uso*: La contribución de las herramientas orientadas a objetos a los sistemas interactivos modernos y especialmente a sus interfaces de usuario es bien conocida. Proporcionan una herramienta potente para poder realizar entornos visuales sencillos y fáciles de manejar para todo tipo de usuarios.

- *Eficiencia*: En primera instancia, el poder extra de las técnicas orientadas a objetos parece pasarnos factura en cuanto al rendimiento de las aplicaciones. En cambio, el utilizar componentes reutilizables de alta calidad, a menudo conlleva mejoras significativas en el rendimiento.

- *Oportunidad, Economía y Funcionalidad*: Las técnicas orientadas a objetos permiten a aquellos que las dominan producir software más rápidamente y a menor coste; facilitan la adición de funciones y pueden en sí mismas sugerir nuevas funciones a añadir.

Son estas fundamentalmente las causas que nos han llevado a seleccionar un lenguaje de programación orientado a objetos.

El lenguaje de programación elegido es el *Visual C++*, fundamentalmente porque cientos de miles de programadores utilizan C++ en, básicamente, cada dominio de aplicación. Esta utilización se apoya en una docena de implementaciones independientes, cientos de bibliotecas, cientos de libros de texto, multitud de revistas técnicas, conferencias etc. Es decir, disponemos de información a diversos niveles. Las primeras aplicaciones implementadas en este lenguaje tenían una fuerte inclinación a la programación de sistemas. Por ejemplo se han escrito varios sistemas operativos importantes en C++. Esto permite utilizar C++ para escribir controladores de dispositivos y otro software que se apoya en la manipulación

directa del hardware bajo restricciones de tiempo real. C++ se diseñó de tal forma que cada característica del lenguaje es utilizable en código que se encuentre bajo severas restricciones de espacio y de tiempo. La mayoría de las aplicaciones tienen secciones de código que son críticas para un rendimiento aceptable. Sin embargo, la mayor parte del código no se encuentra en tales secciones. Para la mayoría del código, el mantenimiento, la facilidad de extensión y facilidad de verificación es clave. El soporte de C++ de estos aspectos ha llevado a un amplio uso allá donde la fiabilidad es una exigencia y en áreas donde los requisitos cambian significativamente a lo largo del tiempo. La estabilidad, la compatibilidad y la escalabilidad son aspectos que caracterizan a este lenguaje de programación. Además, aunque C++ no se diseñó específicamente para el cálculo numérico, este lenguaje posee una gran potencia de cálculo a la hora de resolver problemas científicos y de ingeniería.

Por otra parte los gráficos y las interfaces de usuario son áreas en las cuales C++ ofrece un gran abanico de posibilidades. Cabe destacar también la capacidad que presenta para utilizarse de forma eficaz en aplicaciones que requieren trabajo en diversas áreas de aplicación. Como ejemplo podemos citar las aplicaciones que se han desarrollado en este proyecto fin de carrera, que hacen uso de la red de área local, realizan cálculos numéricos, tratamiento de formatos gráficos, interacción con el usuario etc. Tradicionalmente estos campos de aplicación se han considerado distintos y con frecuencia han pertenecido a distintas comunidades técnicas, utilizando diversos lenguajes de programación.

Finalmente y a modo de introducción del lenguaje, diremos que en este lenguaje de programación orientada a objetos, la modularidad anteriormente descrita se consigue llevar a su máxima expresión mediante el concepto de clase. Una clase en C++ es un tipo. Es decir, especifica cómo se comportan los objetos de su clase: cómo se crean, cómo se pueden manipular y cómo se destruyen. Una clase puede además especificar cómo se representan los objetos. La clave para escribir buenos programas es diseñar las clases de modo que cada una represente claramente un único concepto. Pero un concepto no existe en el vacío, siempre hay grupos de conceptos relacionados. La organización de las relaciones entre las distintas clases de un programa suele ser más difícil que situarlos en primer lugar en clases individuales. Una de las herramientas intelectuales más poderosas para gestionar la

complejidad es la ordenación jerárquica. Es decir, la organización de los conceptos relacionados en una estructura de árbol, con el concepto más general como raíz. En C++, las clases derivadas representan dichas estructuras.

Hemos de reseñar también como motivación a la hora de elegir Visual C++ como lenguaje de programación para implementar este proyecto fin de carrera, su futura integración formando parte de otras aplicaciones desarrolladas en el Departamento de Ingeniería de Sistemas y Automática en este mismo lenguaje. Entre ellas la aplicación de detección de humo que trabaja con el posicionador que es uno de los destinos finales de este trabajo.

### 2.6.2 DESARROLLO DEL SOFTWARE

Una vez decidido el lenguaje de programación sobre el que vamos a realizar el trabajo, hemos tenido que proponer una estructura o secuencia lógica de ejecución. Es decir, cuando se va a abordar un proyecto de una magnitud considerable, no se puede empezar a escribir código y a probar si funciona o no. Se han de seguir ciertas pautas de comportamiento, una organización que nos facilitará el trabajo de manera significativa, además de llevarnos a la solución adecuada por un camino más corto, produciéndose una optimización en el tiempo de trabajo.

Para tener éxito al diseñar y construir un software se necesita disciplina, es decir, necesitaremos dar al proceso de desarrollo un enfoque de ingeniería. Pero existe un problema a la hora de realizar el trabajo y es que aunque gestores y profesionales reconocen la necesidad de dar un enfoque más disciplinado, continúan debatiendo sobre la manera en que se va a aplicar esta disciplina. Muchas compañías y profesionales del software todavía desarrollan software de manera algo peligrosa, incluso cuando construyen sistemas para dar servicio a las tecnologías más avanzadas de hoy en día. El no conocer los nuevos métodos de desarrollo producen como resultado software de mala calidad.

La ingeniería del Software es una tecnología multicapa con la siguiente estructura:

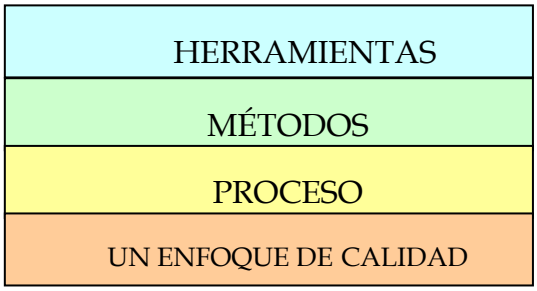



**Figura 2.3. Distribución en capas de la Ingeniería del Software.**

Cualquier enfoque de ingeniería debe apoyarse sobre un compromiso de organización de calidad. El fundamento de la ingeniería del software es la capa de proceso. Este proceso es la unión que mantiene juntas las capas de tecnología y que permite un desarrollo racional y oportuno de los proyectos software. El proceso define un marco de trabajo para un conjunto de *áreas claves de proceso* que se deben establecer para la entrega efectiva de la tecnología de la ingeniería del software. Las áreas claves del proceso forman la base del control de gestión de proyectos software y establece el contexto en el que se aplican los métodos técnicos, se obtienen productos del trabajo, se establecen hitos, se asegura la calidad y el cambio se gestiona adecuadamente.

Los métodos de ingeniería del software indican cómo construir técnicamente las aplicaciones. Estos métodos abarcan una amplia gama de tareas que incluyen análisis de requisitos, diseño, construcción de programas, pruebas y mantenimiento. Además dependen de un conjunto de principios básicos que gobiernan cada área de la tecnología e incluyen actividades de modelado y otras técnicas descriptivas.

Las herramientas de la ingeniería del software proporcionan un enfoque automático o semiautomático para el proceso y para los métodos. Cuando se integran herramientas para que la información creada por una herramienta la pueda utilizar otra, se establece un sistema de soporte para el desarrollo del software llamado *ingeniería del software asistida por ordenador*.

De esta forma podemos afirmar que el trabajo que se asocia a la ingeniería del software se puede dividir en tres fases genéricas, con independencia de la aplicación, tamaño o complejidad del proyecto. Cada fase se encuentra con una o varias cuestiones de las destacadas anteriormente. Pasamos a continuación a describir estas fases del trabajo que hemos seguido fielmente a la hora de llevar a cabo este proyecto fin de carrera:

- La fase de *definición* se centra en el qué. Es decir, durante la definición, el programador intenta identificar qué información ha de ser procesada, qué función y rendimiento se desea, qué comportamiento del sistema, qué interfaces van a ser establecidas, qué restricciones del diseño existen y qué criterios de validación se necesitan para definir un sistema correcto. Por tanto han de identificarse los requisitos clave del sistema y del software. Aunque los métodos aplicados durante la fase de definición pueden variar dependiendo del paradigma de ingeniería del software, para el desarrollo de este proyecto hemos contemplado las siguientes tareas principales:
  - Estudio de los sistemas de información.
  - Planificación del proyecto software.
  - Análisis de los requisitos.
- La fase de *desarrollo* se centra en el cómo. Es decir, durante el desarrollo un ingeniero de software intenta definir cómo han de diseñarse las estructuras de datos, cómo ha de implementarse la función dentro de una arquitectura de software, cómo han de implementarse los detalles procedimentales, cómo han de caracterizarse los interfaces, cómo ha de traducirse el diseño en un lenguaje de programación y cómo ha de realizarse la prueba. Los métodos aplicados durante la fase de desarrollo de este proyecto fin de carrera han sido:
  - Diseño del software.
  - Generación del código.
  - Prueba del software.
- La fase de mantenimiento se centra en el cambio que va asociado a la corrección de errores, a las adaptaciones requeridas a medida que evoluciona el entorno del software y a cambios en las mejoras producidas por los requisitos

cambiantes de las especificaciones. Durante la fase de mantenimiento se distinguen cuatro tipos de cambio:

- *Corrección*: Incluso llevando a cabo las mejores actividades de garantía de calidad, es muy probable que descubramos defectos en el software. El mantenimiento correctivo cambia el código para corregir los defectos.

- *Adaptación*: Con el paso del tiempo es probable que cambie el entorno original (por ejemplo: CPU, Sistema operativo,...) para el que se desarrolló el software. El mantenimiento adaptativo produce modificación en el código para acomodarlo a los cambios de su entorno externo. En nuestro caso ha sido necesaria esta adaptación en la fase de integración del proyecto en la aplicación de detección de humo, para lo que hemos tenido que hacer modificaciones en las aplicaciones. Además como el equipo que se encuentra instalado en el laboratorio de la planta baja posee características distintas en cuanto a hardware se han hecho consideraciones distintas a la hora de implementar los programas.

- *Mejora*: Conforme se ha ido probando el software hemos descubierto funciones adicionales que pueden producir beneficios. El mantenimiento perfectivo lleva a las aplicaciones más allá de sus requisitos funcionales originales.

- *Prevención*: El software de los equipos de deteriora debido al cambio y por esto el mantenimiento preventivo, también llamado reingeniería del software, se debe conducir a permitir que el software sirva para las necesidades de los usuarios finales. En esencia, el mantenimiento preventivo hace cambios en aplicaciones a fin de que se puedan corregir, adaptar y mejorar más fácilmente.

Pero para llevar a buen término un proyecto software se debe incorporar al proceso una estrategia de desarrollo que tendrá en cuenta las capas de herramientas mostradas en la figura 2.3 y las tres fases genéricas anteriormente descritas. Esta estrategia a menudo se llama *modelo de proceso* o *paradigma de ingeniería del software.* Se selecciona un modelo de proceso para la ingeniería del software según la naturaleza del proyecto y de la aplicación, los métodos y las herramientas a utilizar y los controles y entregas que se requieren.

Existen muchos modelos de procesos para la ingeniería del software. Cada uno representa un intento de ordenar una actividad inherentemente caótica. Es por ello por lo que cada autor sugiere una estrategia distinta a la hora de desarrollar un proyecto. Dentro de estas líneas de trabajo encontramos:

- El modelo secuencial lineal.
- El modelo de construcción de prototipos
- El modelo DRA (Desarrollo rápido de aplicaciones).
- El modelo incremental.
- El modelo espiral.
- El modelo espiral WINWIN.
- El modelo de desarrollo concurrente.
- Desarrollo basado en componentes.
- El modelo de métodos formales.
- Técnicas de cuarta generación.

Sería un proceso largo y tedioso el detallar los fundamentos de cada una de las técnicas y asimismo no es objetivo de este documento el proponer al lector un método para desarrollar las aplicaciones. El objetivo de esta parte es tratar de situar a la persona interesada en este proyecto lo más cerca posible al punto de vista del alumno que lo ha llevado a cabo y explicarle la solución adoptada desde la citada perspectiva. Es por ellos por lo que remitimos al lector a la bibliografía del proyecto si está interesado en conocer los pormenores de las técnicas citadas anteriormente.

La técnica elegida para llevar a cabo este proyecto fin de carrera es *El Modelo Lineal Secuencial* principalmente por su simplicidad y contrastada efectividad, ya que es el paradigma más antiguo y más extensamente utilizado en la ingeniería del software. También se conoce como *ciclo de vida básico* o *modelo en cascada.* Este sugiere un enfoque sistemático, secuencial, para el desarrollo del software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación pruebas y mantenimiento.

Las actividades que comprende el modelo se muestran en la siguiente figura y se detallan a continuación:

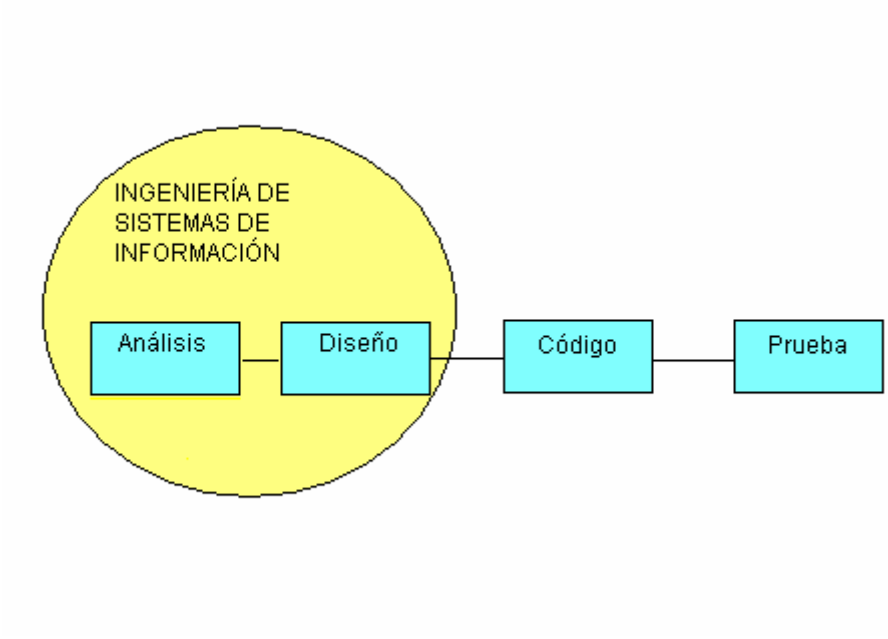



**Figura 2.4. Actividades que comprende el *modelo lineal secuencial*.**

- *INGENIERÍA Y MODELADO DE SISTEMAS DE INFORMACIÓN:* Puesto que este software tiene como uno de sus propósitos la integración en un sistema mayor, el trabajo comienza estableciendo los requisitos de todos los elementos del sistema y asignando al software un subgrupo de estos requisitos. Esta visión del software es esencial cuando este se debe interconectar con otros elementos como son hardware y personas. La ingeniería y el análisis de sistemas comprende los requisitos que de recogen en el nivel del sistema con una pequeña parte del análisis y el diseño.

- *ANÁLISIS DE LOS REQUISITOS DEL SOFTWARE:* El proceso de reunión de los requisitos se intensifica y se centra especialmente en el software. Para comprender la naturaleza de los programas a construirse, el ingeniero de software debe comprender el dominio de información del software, así como la función requerida, comportamiento, rendimiento e interconexión. Los requisitos de las aplicaciones que se han desarrollado en este proyecto han sido descritos previamente en el apartado 2.5. Estos requisitos han sido impuestos por el entorno donde se implantarán las aplicaciones (red de área local instalada en el laboratorio y equipos) así como por las necesidades específicas de los proyectos donde se integrarán (proyecto de detección de humo).

- *DISEÑO:* El diseño del software es realmente un proceso de muchos pasos que se centra en cuatro atributos distintos de programa:
  - Estructura de datos.
  - Arquitectura de software.
  - Representaciones de interfaz.
  - Detalle procedimental (algoritmo).

  Este proceso de diseño traduce requisitos en una representación del software donde se pueda evaluar la calidad antes de que comience la codificación.

- *GENERACIÓN DEL CÓDIGO:* El diseño se debe traducir en una forma legible por la máquina. El paso de generación del código lleva a cabo esta tarea. Si se lleva a cabo el diseño de una forma detallada, la generación del código se realiza de una forma mecánica.

- *PRUEBAS:* Una vez que se ha generado el código, comienzan las pruebas del programa. El proceso de prueba se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado y en los procesos externos funcionales; es decir, realizar las pruebas para la detección de errores y asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos. Los resultados de estas pruebas se detallan en el capítulo 9, dedicado a tal efecto de este documento.

- *MANTENIMIENTO:* El software indudablemente sufrirá cambios una vez implementado en el sistema final donde se va a integrar. Estos cambios se producirán porque se han detectado errores, porque el programa debe adaptarse para acoplarse a los cambios de su entorno externo o porque el usuario final requiere mejoras funcionales o de rendimiento. El soporte y mantenimiento del software vuelve a aplicar cada una de las fases precedentes a un programa ya existente y no a uno nuevo.

Por tanto, una vez llegados a este punto y a modo de resumen podemos decir que cuando se lleva a cabo un proyecto software no debemos empezar el trabajo de cualquier manera. Es de vital importancia seguir método de ejecución estructurado de las actividades para optimizar el rendimiento en el trabajo y obtener de esta forma

resultados satisfactorios en un tiempo adecuado. Esta metodología producirá también mejores resultados en las aplicaciones desarrolladas.

### 2.6.3 DOCUMENTACIÓN DEL SOFTWARE

A la hora de realizar la documentación relativa a las aplicaciones desarrolladas, hemos tenido en cuenta varios factores entre los que destacan la claridad, simplicidad y la suposición de que el lector tiene al menos unas nociones básicas sobre el lenguaje de programación en el que están escritos los códigos de los programas (Visual C++). Distinguimos tres tipos de documentación:

- La *documentación externa*, que permite a los usuarios conocer la potencia del sistema y usarlo convenientemente. Esta es una consecuencia de la definición de la facilidad de uso. Esta documentación externa es la que tiene el lector actualmente en sus manos y que describe no sólo la funcionalidad de los distintos módulos que componen el trabajo sino que además detallan el interfaz gráfico y otras características de las aplicaciones.

- La *documentación interna*, que permite a los programadores comprender la estructura e implementación de un sistema. Esta es una consecuencia del requisito de extensibilidad. En este proyecto este tipo de documentación se proporciona a través de este documento y a través de los comentarios añadidos en el código fuente que aclaran en todo momento la utilidad de las distintas funciones y variables empleadas en el programa.

- Por último está la documentación referente a las interfaces de los distintos módulos (o clases en el caso que nos ocupa), que permitirá a los programadores que pretendan reutilizarlos en sus aplicaciones comprender las funciones proporcionadas por el módulo sin tener que entender su implementación. La documentación referente a interfaces de este proyecto se encuentra dividida entre los comentarios intercalados en el código fuente y los capítulos posteriores de este documento.

En lugar de tratar la documentación como un producto propio del software, es preferible generar un código lo más autodocumentado posible. Esto se aplica a los tres tipos de documentación:

- Incluyendo facilidades de ayuda en línea.

- Un buen lenguaje de implementación puede eliminar muchas de las necesidades de documentación interna, si favorece la claridad y la estructura. Este será uno de los requisitos fundamentales de la notación orientada a objetos que se desarrolla a lo largo de este proyecto.

- La notación soportará la ocultación de información y otras técnicas (tales como aserciones) para separar la interfaz de los módulos de su implementación. Sería posible entonces utilizar herramientas para producir automáticamente documentación de la interfaz del módulo a partir del texto de los módulos.

Por estos motivos es por lo que consideramos adecuada la solución adoptada a la hora de incluir la documentación relativa a los programas.

### 2.6.4 SOCKETS

En este capítulo de este documento, donde se detallan los motivos que han llevado al alumno a tomar las decisiones que han dado como resultado este proyecto fin de carrera, no podemos dejar a un lado la elección del *socket* como vía de comunicación.

En el capítulo 4 de esta memoria se describe detalladamente en qué consiste un socket y la forma de trabajar con él, dentro del marco de las comunicaciones de red que emplean el protocolo Ethernet. En cambio realizaremos en este punto una breve introducción al concepto básico y a las ventajas que presenta esta vía de comunicación.

En todas las áreas de la tecnología se tiende siempre a crear modelos simplificados o niveles que sean transparentes a usuarios de aplicaciones superiores. Con esta premisa surgió el paradigma "socket" popularizado por la Berkeley Software Distribution (BSD) de la Universidad de California, en Berkeley. Este socket, o enchufe, consiste en un conjunto de órdenes para gestionar la transmisión de datos en cualquier aplicación o programa, pero a diferencia de lo ocurrido hasta entonces donde cada programador se hacia las suyas propias, se trata de un conjunto de órdenes estándar, común para todos los usuarios del entorno para el que es creado. Cuando hablamos de Windows Sockets nos referimos a un interfaz de red, programado para usar en Microsoft Windows. Este interfaz posee el característico estilo de las rutinas socket de Berkeley y las extensiones especificas de Windows

diseñadas para permitir al programador obtener ventaja de la forma de conducir los mensajes que tiene Windows.

La especificación Windows Sockets sirve para proporcionar una sencilla API (Interface de Programación de Aplicaciones, herramienta que mejora y potencia los servicios de Windows) con la que los programadores pueden desarrollar aplicaciones y utilizar los diversos sistemas de software de red tanto comerciales como estándar. Además, en el contexto de una versión particular de Microsoft Windows, Sockets define un modo de conectarse (ABI) tal que una aplicación escrita para el Windows Sockets API pueda trabajar con un protocolo correspondiente facilitado por cualquier vendedor de software de red.

Con este conjunto de órdenes el programador de aplicaciones no tiene porque preocuparse del nivel de red, tan solo sabe que haciendo uso de las llamadas a subrutinas ya hechas y siguiendo el estándar establecido la transmisión de datos se realizara de forma eficaz y correcta.

Podemos decir que los sockets no son más que puntos o mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando estos procesos en distintas máquinas. Un socket es al sistema de comunicación entre ordenadores lo que un buzón o un teléfono es al sistema de comunicación entre personas: un punto de comunicación entre dos agentes (procesos o personas respectivamente) por el cual se puede emitir o recibir información.

La comunicación entre procesos a través de sockets se basa en la filosofía *CLIENTE-SERVIDOR*: un proceso en esta comunicación actuará de *proceso servidor* creando un socket cuyo nombre conocerá el *proceso cliente*, el cual podrá "hablar" con el proceso servidor a través de la conexión con dicho socket nombrado. Esta características hace que este tipo de comunicación sea ideal para implantarlo en nuestro proyecto fin de carrera, donde hemos de implementar un software basado en el modelo cliente-servidor.

Además, la comunicación a través de socket presenta las siguientes características:

- Fiabilidad de transmisión.

- Mantenimiento del orden de los datos.
- No duplicación de los datos.
- El "Modo Conectado" en la comunicación.
- Envío de mensajes urgentes.

Como se puede inferir de la descripción de los sockets, estos presentan un método sencillo a la vez que eficaz para llevar a cabo conexiones de red. Además, el lenguaje de programación Visual C++, ofrece un potente interfaz para poder incluir las funciones relativas a la transmisión vía socket en las aplicaciones desarrolladas. La documentación relativa a sockets que ofrece Visual C++ es amplia y completa, lo que ha facilitado en gran medida el trabajo a la hora de realizar el diseño.

Por tanto todo el conjunto presenta un alto grado de integración y esto junto con las demás ventajas expuestas ha sido la razón por la que se ha realizado el diseño basado en este tipo de comunicación.

### 2.6.5 COMPRESIÓN DE IMÁGENES

Otra de las decisiones que han marcado el desarrollo del trabajo ha sido la de incluir un algoritmo de compresión para las imágenes transmitidas. El objetivo de este algoritmo sería el de descargar en la medida de lo posible la red por la que se realiza la transmisión para agilizar el rendimiento en casos de tráfico masivo en los casos en que tengamos una gran demanda de imágenes por parte del Cliente.

Pero esta compresión, como hemos detallado en el capítulo 5, ha de estar sujeta a estrictas especificaciones. Estas especificaciones se resumen las siguientes:

- La compresión ha de ser sin pérdidas dado el contenido de la información que se va a transmitir en las aplicaciones donde se va a integrar este proyecto.
- El algoritmo de compresión debe funcionar muy rápido tanto en la compresión como en la descompresión, puesto que el tiempo de transmisión varía poco para los tamaños de imagen que se suelen manejar y no se trata de hacer el sistema excesivamente lento por culpa de este proceso.
- Las tasas de compresión deben ser útiles.

Teniendo en cuenta la anterior y remitiendo al lector al capítulo 5 del proyecto, diremos que tras haber realizado un proceso de documentación, implementación y

prueba de los algoritmos de compresión sin pérdidas más extendidos, se ha tomado la decisión de emplear el algoritmo de Huffman Adaptativo para realizar este paso.

Es de vital importancia reseñar en este punto que en este proyecto fin de carrera se propone la compresión de la información transmitida como una opción para el usuario. Es decir, se han desarrollado aplicaciones Cliente y Servidor con compresión y sin compresión de los datos, con el fin de que sea el usuario final el que decida la aplicación que desea implementar. El motivo que nos han llevado a plantearlo así es porque bajo determinadas circunstancias la compresión de los datos pudiera llegar a ser innecesaria. Estos hechos son los siguientes:

- Puesto que la compresión de los datos requiere un fuerte procesamiento software, cuando nos encontramos trabajando con equipos de bajas prestaciones, el proceso de compresión podría ralentizar en exceso el proceso de la transmisión.

- En entornos donde la red se encuentra poco cargada, no necesitamos preocuparnos por la mayor o menor carga que nosotros añadamos, con el fin también de agilizar los procesos de transmisión y emplear las aplicaciones Cliente y Servidor más simples.

Por otra parte, el uso de las aplicaciones que incluyen la compresión de imágenes se hace especialmente recomendable:

- En aquellos entornos donde exista una red altamente cargada y sea importante la carga que nosotros añadamos. Tal y como se verá en el capítulo final de este proyecto, la compresión de la información puede suponer a la larga un ahorro de ancho de banda de transmisión considerable.

- Las pruebas realizadas en equipos con tecnología actual nos han indicado que el coste en tiempo de proceso del algoritmo de compresión implementado es insignificante. Esto supone que el ahorro de ancho de banda de la transmisión, aún sin ser expresamente necesario, no nos implica ningún coste adicional.

- Cuando las aplicaciones con compresión se implantan en equipos de última generación y la red sobre la que funcionan se encuentra sobrecargada, se produce un beneficio en los tiempos de funcionamiento, lo que hace muy aconsejable su utilización.

### 2.6.6 EL FORMATO BMP

Otro de los aspectos importantes que se han tratado en este proyecto fin de carrera ha sido el formato de los archivos que se transmiten.

En principio, el sistema de transmisión mediante sockets implementado no depende del tipo de fichero que se envía, ya que la transmisión se realiza byte a byte, en bloques de tamaño variable. En cambio, los programas desarrollados tienen como objetivo la transmisión de imágenes, por lo que vamos a realizar una breve descripción del formato empleado tanto para la representación como para el almacenaje de dichas imágenes.

Se ha elegido el formato Windows Bitmap por varios motivos entre los que destacan:

- Este tipo de imagen es el de uso más común y el único soportado por la totalidad de programas Windows, así como por el propio sistema operativo.
- Es un formato perfectamente integrado en el lenguaje de programación Visual C++ (en el que se ha implementado este proyecto), que ofrece objetos y un gran abanico de funciones para manipular este tipo de ficheros.
- La mayoría de las tarjetas digitalizadoras permiten guardar las imágenes en este formato.
- Todas las imágenes con las que trabajan los proyectos en los que se pueden emplear estas aplicaciones están en formato BMP.
- Los archivos Windows Bitmap se almacenan en un mapa de bits independiente del dispositivo; *DIB* (Device-Independent Bitmap), lo que permite a Windows representar dicho mapa de bits en cualquier dispositivo de salida. Independiente del dispositivo significa que el color de píxel se especifica de forma independiente al método usado para representar el color.
- Carece de compresión. Esto implica que al salvar una imagen en formato BMP tenemos la seguridad de que va a ser idéntica al original, cosa que no podemos decir de aquellos formatos como el JPEG, que incluso tiene una cierta componente aleatoria en lo que sería la variación sobre el original. Existen formatos de compresión sin pérdida, pero por ser menos conocidos o estar optimizados para menos profundidad de color (Gif), no se han tenido en cuenta.

En este formato de archivo las imágenes se representan píxel por píxel, con una profundidad de color de 24 bits (color real). También es posible ( y además es la situación más frecuente) usar una paleta limitada, lo que permite usar entre 1 y 16 bits para representar el color de cada píxel. Teóricamente es posible comprimir los datos mediante un par de algoritmos basados en el RLE (Run Length Encoding). Pero muy pocos editores de archivos BMP soportan este formato comprimido. Por ello esta capacidad es muy poco empleada y no vamos a profundizar más en este aspecto. El formato BMP permite almacenar un solo fichero en cada fichero, por lo que no admite animaciones ni efectos de este tipo.

En cuanto a la estructura de estos archivos diremos que están compuestos por cuatro partes perfectamente diferenciadas:

- *Cabecera de Archivo (File Header)*: Contiene información básica para identificar el formato del archivo y para recuperar las diferentes estructuras que se encuentran presentes en dicho fichero. Dentro de la cabecera de archivo se distinguen los siguientes campos:
  - *bfType (2 bytes)*: La identificación del archivo. Contiene la cadena ASCII "BM" y todo fichero BMP comienza con estos dos bytes.
  - *bfSize (4 bytes)*: Indica el tamaño del archivo completo en bytes.
  - *bfReserved1 (2 bytes)*: Reserva para futura expansiones.
  - *bfReserved(2 bytes)*: Reserva para futura expansiones.
  - *bfOffBits (4 bytes)*: La distancia entre esta estructura y la estructura de datos en bytes.
- *Cabecera de información de la imagen (Info Header)*: Contiene la información sobre la imagen, como el tamaño, número de colores, compresión... Los campos que nos vamos a encontrar en la estructura definida en C son:
  - *biSize (4 bytes)*: Indica el tamaño de esta estructura. Inicialmente parece de poca utilidad pero es importante para futuras versiones.
  - *biWidth (4 bytes), biHeight (4 bytes)*: Anchura y altura en píxeles de la imagen.

- *biPlane (2 bytes)*: Indica el número de planos de la imagen. BMP solamente soporta un plano.
- *biBitCount (2 bytes)*: Es el número de bits por píxel (1, 4, 8 ó 24).
- *biCompression (4 bytes):* Indica el tipo de compresión aplicada. Puede ser BI_RGB (sin compresión), BI_RLE8 (se trata de compresión RLE en una base de 8 bits) o BI_RLE4 ( que es el mismo caso que el anterior pero en una base de 4 bits).
- *biSizeImage (4 bytes)*: Indica el tamaño en bytes de la estructura de datos.
- *biXPelsPerMeter (4 bytes), biYPelsPerMeter (4 bytes)*: Indica la densidad de píxel de la imagen real en ambos ejes.
- *biClrUsed (4 bytes)*: Indica cuantos colores hay en la paleta.
- *BiClrImportant (4 bytes)*: Es el número de colores que se necesitan para representar una imagen. Si vale cero entonces se deben usar todos los colores.

- *Paleta de colores*: Cuando el campo biBitcount de la cabecera de información de la imagen no está configurado a 24, entonces se indica una paleta. Esta es en realidad una tabla donde a cada código entre 0 y $2^{biBitCount-1}$ corresponde un color particular definido en el estilo RGB. Esto significa que hay dos entradas para 1 bit, 16 entradas para 4 bits y 256 para 8 bits. Cada color se especifica mediante cuatro bytes. El primero representa la componente roja, el segundo la componente verde y el tercero la azul. El cuarto byte está actualmente indefinido. El número asignado a cada color es muy importante; de hecho, cuando un dispositivo de salida no puede representar todos los colores (por ejemplo cuando tenemos un bitmap de 256 colores y una tarjeta de video de 16 colores) se empieza por representar los primeros de la paleta. Por esta razón, los colores más importantes (normalmente los más frecuentes) se deben almacenar en las primeras posiciones con el fin de mejorar el rendimiento.

- *Datos*: La última y más grande de las partes de un BMP es la parte de los datos. La imagen es barrida línea a línea, de abajo a arriba. Para cada línea los píxeles están ordenados de izquierda a derecha. Esto significa que el primer byte de la tabla será el píxel de la esquina inferior izquierda de la imagen y el último

byte representa el píxel de la esquina superior derecha. De cada píxel se encuentra representado su código de color. En particular, si biBitCount es 1, por cada byte habrá almacenado 8 píxeles, si biBitCount es 4, se almacenarán dos píxeles por cada byte, si es 8 representaremos un byte por píxel. Finalmente si biBitCount es 24 entonces tendremos una imagen de color real y a cada píxel corresponderán 3 bytes. El primero de los bytes indica la componente azul, el segundo la verde y el tercero la roja (BGR no RGB). Además cada línea debe tener un número de bits que debe ser divisible por 32. Esto implica que si no es así se rellena con ceros hasta que se alcanza un valor admisible. El número de bytes extras que hay que añadir para el relleno es:

$$\text{Bytes Extras} = (4 - (3*\text{AnchuraImagen})/4)/4$$


**Figura 2.5. Estructura del archivo BMP.**

El orden de estas estructuras debe respetarse. La única excepción es la paleta, que no se debe incluir si y sólo si la imagen es de color real (16 millones de colores).

# CAPÍTULO 3

# REDES DE ÁREA LOCAL

# 3 REDES DE ÁREA LOCAL

## 3.1 INTRODUCCIÓN

En este capítulo trataremos de abordar de forma meramente conceptual aquellos conocimientos básicos sobre redes y más concretamente sobre redes de área local aplicados durante la realización del proyecto. Tendrá además como objetivo acercar este documento a aquellas personas interesadas en él y que no tienen un gran conocimiento en la materia. Es por ello por lo que considero oportuno hacer la siguiente descripción.

## 3.2 REDES DE EQUIPOS

En su nivel más elemental, una red de equipos consiste en dos equipos conectados entre sí con un cable que les permite compartir datos. Todas las redes de equipos, independientemente de su nivel de sofisticación, surgen de este sistema tan simple.

Dichas redes de equipos aparecen como respuesta a la necesidad de compartir datos de forma rápida. Los ordenadores personales son herramientas potentes que permiten procesar y manipular rápidamente grandes cantidades de datos, pero no permiten que los usuarios compartan información de forma eficiente. De hecho, antes de la aparición de las redes, los usuarios necesitaban imprimir sus documentos o copiar archivos de documentos en un disco para que otras personas pudieran editarlos o utilizarlos. Si otras personas realizaban modificaciones en el documento, no existía un método fácil para combinar los cambios. Resulta evidente que estos métodos son demasiado lentos e ineficientes para cubrir las necesidades y expectativas de los usuarios informáticos de hoy día.

## 3.3 REDES DE ÁREA LOCAL

Una red local es un sistema de interconexión entre ordenadores que permite compartir recursos e información. Para ello es necesario contar, además de con los ordenadores correspondientes, con las tarjetas de red, los cables de conexión, los dispositivos periféricos y el software conveniente.

Según su ubicación se pueden distinguir varios tipos de redes en función de su extensión.

- Si se conectan todos los ordenadores dentro de un mismo edificio, se denomina *LAN (Local Area Network)* .
- Si se encuentran en edificios diferentes distribuidos dentro de la misma universidad, se denomina *CAN(Campus Area Network)*.
- Si se encuentran en edificios diferentes distribuidas en distancias no superiores al ámbito urbano*, MAN (Metropolitan Area Network)*.
- Si están instalados en edificios diferentes dentro de la misma o distinta localidad, provincia o país, *WAN (Wide Area Network)*.

Según la forma en que estén conectados los ordenadores, se pueden establecer varias categorías:

- Aquellas redes que utilizan enlaces a través de puertos serie o paralelo para transferir archivos o compartir periféricos se denominan *Redes sin tarjetas*.
- *Redes punto a punto* son aquellas que hacen posible la comunicación entre dos ordenadores determinados de forma permanente.
- Denominamos *Redes entre iguales* a aquellas en las que todos los ordenadores conectados pueden compartir información con los demás.
- Por último están las *Redes basadas en servidores* centrales utilizando el modelo básico cliente servidor.

## 3.4 VENTAJAS DE LA UTILIZACIÓN DE REDES LOCALES

Las principales ventajas del empleo de redes locales se pueden resumir en los siguientes puntos:

- *Compartir información (o datos)*: La capacidad de compartir información de forma rápida y económica ha demostrado ser uno de los usos más populares de la tecnología de redes. Éstas logran que prácticamente cualquier tipo de dato esté disponible simultáneamente para cualquier usuario que lo necesite. Las redes enlazan también a las personas proporcionando una herramienta efectiva para la

comunicación a través de correo electrónico. Los mensajes se envían instantáneamente a través de la red, los planes de trabajo pueden actualizarse tan pronto como ocurran cambios y por ejemplo, pueden planificarse las reuniones sin necesidad de llamadas telefónicas. Esto reduce o elimina por completo la duplicidad de trabajo.

- *Compartir Hardware y Software*: Antes de la aparición de las redes, cada usuario necesitaba su propia impresora, trazadores y otros periféricos. Muchos de estos dispositivos tienen un coste elevado en el mercado y por tanto la solución anterior es bastante poco económica. Con la introducción de las redes, nos basta en la mayoría de los casos con tener un solo dispositivo que comparten varios usuarios que se conectan al mismo mediante la infraestructura de red de la que venimos hablando. Además dicha infraestructura puede utilizarse para compartir y estandarizar aplicaciones, como tratamiento de texto, hojas de cálculo etc. Y así asegurarse de que todas las personas de la red utilizan las mismas aplicaciones y las mismas versiones de estas aplicaciones. De esta forma se facilita que los usuarios aprendan a usar bien una determinada aplicación antes que intentar aprender cuatro o cinco aplicaciones distintas.

- *Centralización de la administración y el soporte*: Para el personal técnico es mucho más eficiente dar soporte a una versión de un sistema operativo o aplicación y configurar todos los equipos de un mismo modo que dar soporte a muchos sistemas y configuraciones individuales diferentes. Además permite mejorar la seguridad y control de la información que se utiliza permitiendo la entrada de determinados usuarios, accediendo únicamente a cierta información o impidiendo la modificación de diversos datos.

En definitiva podemos concluir que las redes aumentan la eficiencia y reducen los costes.

## 3.5 CONFIGURACIÓN DE LAS REDES

Las redes según su configuración se dividen en dos categorías fundamentalmente: *Redes peer-to-peer* y *Redes basadas en servidor.*

La diferencia entre ambos tipos de redes es importante, ya que cada tipo presenta distintas capacidades. El tipo de red seleccionado para su instalación dependerá de

factores tales como el tamaño de la organización, en nivel de seguridad requerido, el nivel de soporte administrativo disponible, la cantidad de tráfico de la red etc.

### 3.5.1 REDES PEER-TO-PEER

En este tipo de red no hay servidores dedicados y no existe una jerarquía entre los equipos. Todos los equipos son iguales, y por tanto son pares (*peers*). Cada equipo actúa como cliente y como servidor, y no hay un administrador responsable de la red completa. El usuario de cada equipo determina los datos de dicho equipo que van a ser compartidos en la red. Las redes peer-to-peer se llaman también *grupos de trabajo* o *workgroups*.

En cuanto a coste, las redes con esta configuración son relativamente simples, puesto que no hay necesidad de un potente ordenador central o de los restantes componentes de una red de alta capacidad. Además el software de red no requiere el mismo tipo de rendimiento y nivel de seguridad que en una red diseñada con servidores dedicados.

Por lo mencionado antes es por lo que este tipo de redes son adecuadas en entornos de trabajo donde no existe un gran número de usuarios que comparten recursos, donde la seguridad no es una cuestión fundamental y la organización de la red no va a experimentar un gran crecimiento en un futuro cercano.

### 3.5.2 REDES BASADAS EN SERVIDOR

Cuando el número de usuarios de una red aumenta, dejan de ser adecuadas las redes *peer-to-peer*. Por tanto la mayoría de las redes tienen servidores dedicados. Un *servidor dedicado* es aquel que sólo funciona como servidor y no se utiliza como cliente o como estación. Estos servidores están optimizados para dar servicio con rapidez a peticiones de clientes de la red y garantizar la seguridad de los archivos y directorios. Las redes basadas en servidor se han convertido en el modelo estándar para la definición de redes.

A medida que las redes aumentan su tamaño es habitual que necesiten más de un servidor. Puesto que los servidores necesitan realizar tareas complejas y variadas estos se han especializado para adaptarse a las necesidades de los usuarios. Es por ello por lo que hoy día existen diferentes tipos de servidores incluidos en muchas

redes de gran tamaño, como por ejemplo: *Servidores de archivos* e *impresión, servidores de aplicaciones, servidores de correo, servidores de fax, servidores de comunicaciones* etc.

En cuanto al software tendremos que decir que un servidor de red y su sistema operativo trabajan conjuntamente como una unidad. Es fundamental que el sistema operativo de un elemento servidor pueda obtener el máximo rendimiento del hardware. Por tanto podemos concluir que el software va a desempeñar una tarea muy importante en este tipo de configuración.

Es fácil extraer de todas estas características que la compartición de datos y recursos puede ser administrada y controlada de forma más centralizada. Ello facilita también las tareas de mantenimiento del sistema.

En lo relativo a la seguridad tendremos que decir que es a menudo la razón primaria para seleccionar un enfoque basado en servidor en las redes. En esta topología existe un administrador que define la política y la aplica a todos los usuarios de la red pudiendo gestionar la seguridad.

## 3.6 TOPOLOGÍAS ESTÁNDAR DE REDES

Cuando de habla de topología nos referimos a la organización o distribución física de los equipos, cables y otros componentes de la red.

La topología de una red afecta a sus capacidades y va a tener una importante influencia sobre aspectos tan determinantes como:

- Tipo de equipamiento que necesita la red.
- Las capacidades del equipo.
- El crecimiento de la red.
- Las formas de gestionar la red.

Por este motivo haremos hincapié en que una topología de red requiere planificación. Esta planificación estará enfocada a minimizar los gastos, mejorar la fiabilidad del sistema, evitar tiempos de espera en la transmisión de los datos, permitir un mejor control de la red y facilitar la escalabilidad de la misma.

Vamos a ver a continuación las topologías más utilizadas.

### 3.6.1 CONFIGURACIÓN EN BUS

En esta configuración todas las estaciones de trabajo comparten el mismo canal de comunicaciones, toda la información circula por ese canal y cada una de ellas recoge la información que le corresponde. Este canal es un cable común compartido y se denomina *segmento central*.

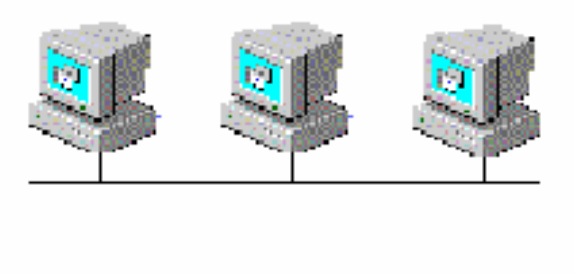


**Figura 3.1. Topología en bus.**

Los equipos conectados de esta forma se comunican enviando datos a un equipo particular, mandando estos sobre el cable en forma de señales electrónicas. La información enviada al canal sólo es recibida o aceptada por aquel equipo cuya dirección coincida con la dirección codificada en la señal original. Como en cada momento sólo puede haber un equipo enviando datos en una red en bus, el número de equipos conectados al bus afectará al rendimiento de la red.

Dado que los equipos que forman parte de este tipo de red o transmiten datos a otros equipos de la red o están escuchando datos procedentes de dichos equipos no son responsables de pasar información al siguiente. Por consiguiente, si falla un equipo esto no afecta al resto de la red. Decimos entonces que se trata de una topología robusta.

Los datos o señal electrónica se envía a toda la red y por tanto viaja de un extremo a otro del cable. Si se permite a la señal que continúe ininterrumpidamente, rebotará una vez y otra por el segmento central y esto evitará que otros equipos envíen señales. Por tanto la señal debe ser detenida una vez que haya tenido la oportunidad de alcanzar la dirección de destino correcta. Para ello se coloca un componente *terminador* en cada uno de los extremos del cable para absorber las

señales libres. Al absorber la señal, el cable se libera para que otros equipos puedan enviar datos. Por tanto todos los extremos de cada segmento de red deben estar conectados a algo, ya sea un equipo, un conector para ampliar la longitud del cable o bien un terminador para evitar que rebote la señal.

En cuanto a la expansión de la red diremos que el cable de la topología en bus puede alargarse ya sea empleando un componente denominado *acoplador* (o *barrel conector*) o bien un dispositivo llamado *repetidor*. El uso de los primeros debe ser limitado puesto que estos conectores debilitan la señal. En cuanto a los segundos no hay ningún problema ya que estos amplifican la señal antes de enviarla por el cable. Eso sí, tendremos que decir que en ninguno de los casos la longitud del cable debe sobrepasar los 2000 metros.

Esta es la configuración más extendida actualmente y es la usada por la red *Ethernet*.

### 3.6.2 CONFIGURACIÓN EN ANILLO

En esta configuración todas las estaciones están conectadas entre sí formando un anillo, de forma que cada estación sólo tiene contacto directo con otras dos. Es decir, los equipos se conectan en un único círculo de cable. A diferencia de la topología en bus, no existen finales con terminadores.

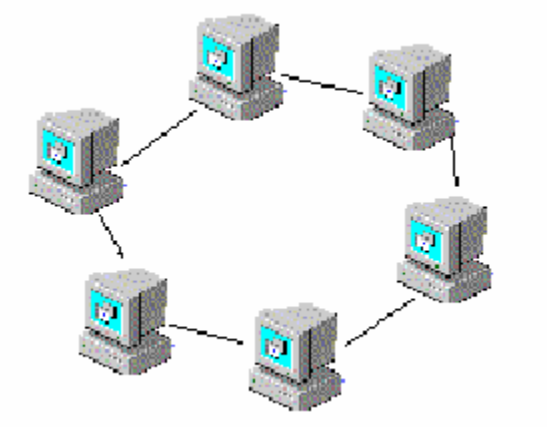


**Figura 3.2. Topología en anillo.**

En las primeras redes de este tipo los datos se movían en una única dirección, de manera que toda la información tenía que pasar por todas las estaciones hasta llegar

a la de destino donde se quedaba. Las redes más modernas disponen de dos canales y transmiten en direcciones diferentes para cada uno de ellos.

La configuración en anillo permite aumentar o disminuir el número de estaciones sin dificultad; pero a medida que aumenta el flujo de información, será menor la velocidad de respuesta de la red.

Un fallo en una estación puede dejar bloqueada la red, pero un fallo en el canal de comunicaciones la dejará bloqueada en su totalidad y además será bastante difícil localizar el fallo y repararlo de forma inmediata.

Su instalación es compleja y su uso está extendido por el entorno industrial. Es usada por la red *Token Ring* de IBM.

### 3.6.3 CONFIGURACIÓN EN ESTRELLA

Esta forma de configuración es una de las más antiguas. Todas las estaciones están conectadas directamente al servidor y todas las comunicaciones se han de hacer necesariamente a través de él.

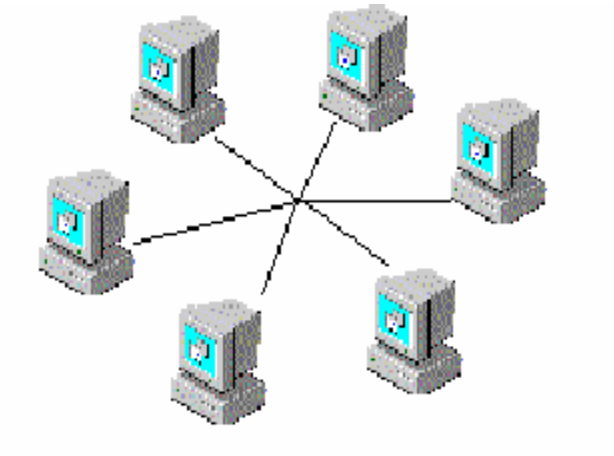


**Figura 3.3. Topología en estrella.**

Ahora podemos incrementar y disminuir fácilmente el número de terminales.

Si se produce un fallo en una de ellos no repercutirá en el funcionamiento general de la red; pero, si se produce un fallo en el servidor, la red completa se vendrá abajo. Por ello podemos decir que es un sistema poco robusto frente a fallos.

Tiene un tiempo de respuesta rápido en las comunicaciones de las estaciones con el servidor central y lento en las comunicaciones entre las distintas estaciones de trabajo.

La red en estrella ofrece la ventaja de centralizar los recursos y la gestión. Sin embargo no es muy conveniente para grandes instalaciones y su coste es caro debido a la gran cantidad de cableado y a la complejidad de la tecnología que se necesita para el servidor.

### 3.6.4 CONFIGURACIÓN EN MALLA

Este tipo de distribución ofrece una redundancia y fiabilidad superiores. Ahora cada equipo está conectado a todos los demás mediante cables separados. Por tanto esta configuración ofrece caminos redundantes por toda la red, de modo que si falla un cable, otro se hará cargo del tráfico. Aunque la facilidad de solución de problemas y la fiabilidad tan elevada son ventajas muy interesantes, estas redes resultan caras de instalar, ya que utilizan muchísimo cableado.

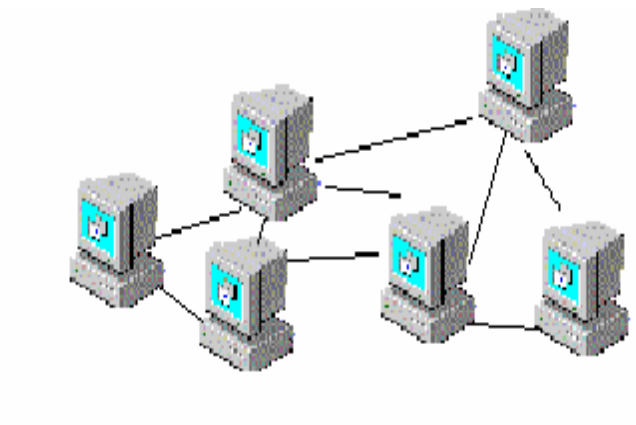


**Figura 3.4. Topología en malla.**

Lo que se hace habitualmente es combinar esta topología con otra formando así una topología híbrida, con el fin de hacer más robustas unas partes del sistema y más económicas otras.

## 3.7 ELEMENTOS DE CONEXIÓN DE UNA RED

Se entiende por elementos de conexión de una red a los cables, tarjetas de red y otros quipos necesarios para conectar entre sí dos ordenadores. Vamos a continuación a hacer una breve descripción de estos elementos.

### 3.7.1 CABLE COAXIAL

Un cable coaxial consta de un núcleo de hilo de cobre rodeado por un aislante, un apantallamiento de metal trenzado y una cubierta externa.

El apantallamiento protege los datos transmitidos absorbiendo las señales electrónicas espúreas, llamadas ruidos, de forma que no pasen por el cable y no distorsionen los datos.

El núcleo de un cable coaxial transporta señales electrónicas que forman los datos. Este núcleo puede ser sólido o de dos hilos.

Rodeando al núcleo hay una capa aislante dieléctrica que la separa de la malla de hilo trenzada que actúa como masa y protege al núcleo del ruido eléctrico y de la intermodulación.

La cubierta exterior no conductora suele ser de goma (Teflón o plástico) y rodea a todo el cable.

Por todo esto podemos decir que el cable coaxial es más resistente a interferencias y atenuación que el cable de par trenzado que ahora describiremos. Es por ello por lo que este tipo de cable es una buena opción para grandes distancias y para soportar de forma fiable grandes cantidades de datos con un equipamiento poco sofisticado.

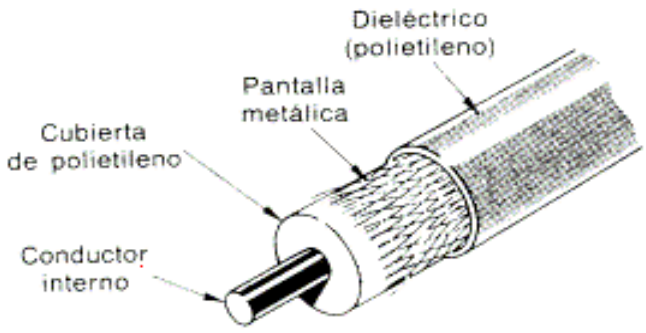


**Figura 3.5. Cable coaxial.**

Hay dos tipos de cable coaxial:

- *Cable thinnet (Ethernet Fino)*: Es un cable flexible y fácil de manejar, por lo que se puede usar para la mayoría de los tipos de instalaciones de redes. Este cable puede transportar una señal hasta una distancia aproximada de 185 metros antes de que la señal comience a sufrir atenuación.

- *Cable thicknet (Ethernet Grueso)*: Es un cable coaxial relativamente rígido. A este tipo de cable se le suele llamar también *Ethernet Estándar* debido a que fue el primer tipo de cable utilizado con esta arquitectura de red. El núcleo de cobre del cable thicknet es más grueso que el del cable thinnet. Cuanto mayor sea el grosor de este núcleo más lejos puede transportar las señales. Por ello este cable puede transportar las señales hasta una distancia de 500 metros antes de que sufran atenuación. Dada la capacidad de este cable a veces se utiliza como enlace central o *backbone* para conectar varias redes más pequeñas basadas en thinnet.

En cuanto al hardware de conexión del cable coaxial diremos que ambos tipos de cable descritos anteriormente usan un componente de conexión llamado *Conector BNC*, para realizar las conexiones entre el cable y los equipos. Existen varios tipos de conectores de la familia BNC como son el conector de cable, el conector en T, el conector acoplador y el conector terminador.

**Figura 3.6. Conector BNC.**

**Figura 3.7. Conector BNCT.**

**Figura 3.8. Acoplador BNC.**

A la hora de decidir qué tipo de cable sería conveniente instalar en nuestro sistema debemos tener en cuenta las siguientes características del cable coaxial:

- Permite la transmisión de voz, video y datos.
- Permite transmitir datos a distancias mayores de lo que es posible con un cableado menos caro.
- Ofrece una tecnología familiar con una seguridad de los datos aceptable.

### 3.7.2 CABLE DE PAR TRENZADO

En su forma más simple el cable de par trenzado consta de dos hilos de cobre aislados y entrelazados.

A menudo se agrupan una serie de hilos de par trenzado y se encierran en un revestimiento protector para formar un cable. El número total de pares que hay en un cable puede variar.

El trenzado elimina el ruido eléctrico de los pares adyacentes y de otras fuentes como motores, relés y transformadores.

Vamos a describir los distintos tipos de cable de par trenzado que podemos encontrar en el mercado:

- *Cable de par trenzado sin apantallar (UTP)*: Este se está convirtiendo en el cableado LAN más utilizado. El segmento máximo de longitud de cable es de 100 metros aproximadamente. En este tipo de cable en número de entrelazados permitidos por metro de cable viene determinado en las especificaciones UTP. Puesto que dicho cable se puede utilizar en una gran variedad de situaciones y construcciones se ofrece una clasificación para asegurar la coherencia de los productos para los clientes. La especificación 568A define las siguientes 5 categorías:
  - *Categoría 1*: Cable telefónico tradicional.
  - *Categoría 2*: Transmisión de datos hasta 4 MBps. Consta de cuatro pares trenzados de hilo de cobre.
  - *Categoría 3*: Transmisión de datos hasta 16 MBps. Consta de cuatro pares trenzados de hilo de cobre.
  - *Categoría 4*: Transmisión de datos hasta 20 MBps. Consta de cuatro pares trenzados de hilo de cobre.
  - *Categoría 5*: Transmisión de datos hasta 100 MBps. Consta de cuatro pares trenzados de hilo de cobre.

  El cable UTP es particularmente susceptible a la intermodulación, pero cuanto mayor sea el número de entrelazados por metro de cable, mayor será la protección frente a las interferencias.

**Figura 3.9. Cable UTP.**

- *Cable de par trenzado apantallado (STP)*: Este cable utiliza una envoltura de cobre trenzado, más protectora y de mayor calidad que la usada en el cable UTP. STP además utiliza una lámina rodeando a cada uno de los pares de hilos. Esto ofrece un excelente apantallamiento en los STP para proteger los datos transmitidos de intermodulaciones exteriores, lo que permite soportar mayores tasas de transmisión que los UTP a distancias mayores.

**Figura 3.10. Cable STP**.

Con respecto al hardware de conexión de los cables de par trenzado diremos que utilizan conectores telefónicos RJ-45 para conectar a un equipo. Este tipo de conector es ligeramente más grande que el conector habitual de teléfono. Además el RJ-45 tiene ocho conexiones de cable mientras que el de teléfono sólo tiene cuatro.

**Figura 3.11. Conector RJ-45.**

En cuanto a la elección de este tipo de cable frente a otro se hará en virtud de las siguientes consideraciones:

- La LAN tienen una limitación de presupuesto.
- Se desea una instalación relativamente sencilla, donde las conexiones de los equipos sean simples.
- No usaremos este cable si necesitamos un gran nivel de seguridad y se debe estar absolutamente seguro de la integridad de los datos.
- No usaremos este cable si los datos se deben transmitir a grandes distancias y a altas velocidades.

### 3.7.3 CABLE DE FIBRA ÓPTICA

En este las fibras ópticas transportan señales digitales de datos en forma de pulsos modulados de luz. Esta es una forma relativamente segura de enviar los datos debido a que, a diferencia de los cables de cobre que llevan los datos en forma de señales electrónicas, los cables de fibra óptica transportan impulsos no eléctricos. Esto significa que el cable de fibra óptica no se puede pinchar y sus datos no se pueden robar.

Dicho cable es apropiado para transmitir datos a velocidades muy altas y con grandes capacidades debido a la carencia de atenuación de la señal y a su pureza.

Una fibra óptica consta de un cilindro de vidrio extremadamente delgado, denominado núcleo, recubierto por una capa de vidrio concéntrica, conocida como

revestimiento. Las fibras a veces son de plástico que es más fácil de instalar pero las distancias de transmisión son menores que en el caso del vidrio.

Puesto que en los filamentos de vidrio las señales viajan en una sola dirección, un cable consta de dos hilos con dos envolturas separadas. Un hilo transmite y el otro recibe. Una capa de plástico de refuerzo alrededor de cada hilo de vidrio y las fibras kevlar ofrecen solidez.

**Figura 3.12. Cable de fibra óptica.**

Las transmisiones por cable de fibra óptica no están sujetas a intermodulaciones eléctricas y son extremadamente rápidas, comúnmente transmiten a unos 100MBps, con velocidades demostradas de hasta 1 gigabit por segundo. Además pueden transportar una señal varios kilómetros.

Utilizaremos este tipo de cableado si:

- Necesitamos transmitir datos a velocidades muy altas y a grandes distancias en un medio muy seguro.
- Si tenemos un presupuesto alto.

### 3.7.4 LA TARJETA DE RED

Estos dispositivos ofrecen la interfaz entre los cables y los equipos. Las funciones de esta tarjeta de red se pueden resumir en los siguientes puntos:

- Preparar los datos del equipo para el cable de red.
- Enviar los datos a otro equipo.
- Controlar el flujo de datos entre el equipo y el sistema de cableado.

- Recibir los datos que llegan por el cable y convertirlos en bytes para que puedan ser comprendidos por la unidad de procesamiento central del equipo (CPU).

En un nivel más técnico podemos decir que la tarjeta de red contiene el hardware y la programación firmware (rutinas software almacenadas en la memoria de sólo lectura, ROM) que implementa las funciones de control de acceso al medio y control de enlace lógico en el nivel de enlace de datos del modelo OSI.

**Figura 3.13. Tarjeta de Red.**

Sobre la adaptación de los datos que realiza la tarjeta de red, podemos decir que este dispositivo se encarga básicamente de transformar la información que viaja por el interior de la CPU a través de buses (transmisión en paralelo) a la estructura de transmisión en serie que es la que soporta el cable de red.

Además de la transformación de los datos, la tarjeta de red tiene que anunciar su localización (o dirección) al resto de la red para diferenciarla de las demás tarjetas. Una comisión del Institute of Electrical and Electronics Engineers (IEEE) asigna bloques de direcciones a cada fabricante de tarjetas de red. Los fabricantes graban las direcciones en los chips de la tarjeta y de esta forma cada una tiene una dirección única en la red.

Debido al efecto que causa en la transmisión de los datos, la tarjeta de red produce un efecto bastante significativo en el rendimiento de toda la red. Si la tarjeta es lenta, los datos no se moverán por la red con rapidez. En una red en bus, donde no se puede utilizar la red hasta que el cable esté libre, una tarjeta lenta puede incrementar el tiempo de espera para todos los usuarios.

En las redes basadas en servidor, dado al alto volumen de tráfico en la red, los servidores deberían estar equipados con tarjetas del mayor rendimiento posible. En cambio, las estaciones de trabajo pueden usar las tarjetas de red más baratas, si las actividades de la red están limitadas a aplicaciones, como puede ser el procesamiento de texto que no generan altos volúmenes de tráfico. Sin embargo, otras aplicaciones como las bases de datos o la ingeniería, se vendrán abajo rápidamente con tarjetas de red inadecuadas.

## 3.8 TRANSMISIÓN DE LA SEÑAL

Se pueden utilizar dos técnicas para transmitir las señales codificadas a través de un cable*: la transmisión en banda base y la transmisión en banda ancha*.

### 3.8.1 TRANSMISIÓN EN BANDA BASE

Este es el método más común dentro de las redes locales. Los sistemas que transmiten en banda base utilizan señalización digital en un único canal. Las señales fluyen en pulsos discretos de electricidad o de luz. Transmiten las señales en forma digital sin emplear técnicas de modulación. En cada transmisión se utiliza todo el ancho de banda y por tanto sólo se puede transmitir una señal simultáneamente. El término *ancho de banda* hace referencia a la capacidad para transferir datos, o a la velocidad de transmisión de un sistema de comunicaciones digital, medido en bits por segundo.

La señal viaja a lo largo del cable de red y, por tanto, gradualmente va disminuyendo su intensidad, y puede llegar a distorsionarse. Por tanto no debe ser excesiva la longitud del citado cable ya que la información puede sufrir variaciones irreversibles.

Como medida de protección, los sistemas en banda base a veces utilizan repetidores para recibir señales y retransmitirlas a su intensidad y definición original. Esto incrementa la longitud útil de un cable.

### 3.8.2 TRANSMISIÓN EN BANDA ANCHA

Consiste en transmitir las señales en forma digital modulando la señal sobre ondas portadoras que pueden compartir el ancho de banda del medio de transmisión mediante multiplexación por división de frecuencia. Estas señales viajan a través del

medio físico en forma de ondas ópticas o electromagnéticas. Con la transmisión en banda ancha el flujo de la información es unidireccional.

Si el ancho de banda disponible es suficiente, varios sistemas de transmisión analógica, como la televisión por cable y transmisiones de redes, se pueden mantener simultáneamente en el mismo cable. A cada sistema de transmisión se le asigna una parte del ancho de banda total. De esta forma se deben configurar todos los dispositivos asociados de forma que solo utilicen las frecuencias que están dentro del rango asignado.

Este método hace imprescindible la utilización de un módem para poder modular y demodular la información. La distancia máxima puede llegar hasta los 50 Km. Los elementos de conexión que se pueden usar son el cable coaxial de banda ancha y el cable de fibra óptica.

## 3.9 MÉTODOS DE ACCESO

Se denomina *método de acceso* al conjunto de reglas que definen la forma en que un equipo coloca los datos en la red y toma los datos del cable. Una vez que los datos se están moviendo por la red, estos métodos de acceso ayudan a regular el flujo de tráfico.

### 3.9.1 CONTROL DE TRÁFICO EN EL CABLE

En una red parece que todo el tráfico se mueve simultáneamente, sin interrupción. Pero nada más lejos de la realidad; lo que ocurre es que los equipos toman turnos para acceder a la red durante breves períodos de tiempo.

Varios equipos pueden compartir el acceso al cable. Sin embargo, si dos equipos tratasen de colocar datos en el cable a la vez, los paquetes de datos de cada equipo podrían colisionar con los paquetes de datos del otro equipo, y ambos conjuntos de paquetes de datos podrían dañarse. Es por esto por lo que si un usuario va a enviar información a otro usuario a través de la red, o se va a acceder a los datos de un servidor, tiene que haber una forma para que los datos puedan acceder al cable sin interferirse entre ellos. Además los métodos de acceso tienen que ser consistentes en la forma de manipular los datos, puesto que si los equipos utilizasen métodos de acceso distintos, la red podría tener problemas, debido a que unos métodos podrían

dominar la ocupación del cable. Al asegurar que un sólo equipo coloca los datos en el cable de red, los métodos de acceso aseguran que el envío y recepción de los datos se realizan de forma ordenada.

### 3.9.2 PRINCIPALES MÉTODOS DE ACCESO

Los tres métodos diseñados para prevenir el uso simultáneo del medio de la red incluyen: métodos de acceso múltiple por detección de portadora (por detección de colisiones o con anulación de colisiones), métodos de paso de testigo que permiten una única oportunidad para el envío de los datos y métodos de prioridad de demandas. Pasamos ahora a describir cada uno de ellos:

- *Método de acceso múltiple por detección de portadora por detección de colisiones (CSMA/CD)*: Con este método, cada uno de los equipos de la red, incluyendo a los clientes y a los servidores, comprueban el cable para detectar el tráfico de la red. Un equipo sólo puede enviar datos cuando detecta que el cable está libre y que no hay tráfico circulando por él. Una vez que el equipo haya transmitido los datos al cable, ningún otro equipo puede transmitir datos hasta que estos hayan llegado a su destino y el medio vuelva a estar libre de nuevo. Si dos equipos tratan de enviar datos en el mismo instante de tiempo, habrá una colisión. Cuando eso ocurre, los dos equipos implicados dejarán de transmitir datos durante un período de tiempo aleatorio y tras este volverán a transmitir la misma información. Cada equipo determina su propio tiempo de espera por lo que es muy difícil que los equipos traten de transmitir de nuevo en el mismo instante de tiempo. La posibilidad de detección de colisiones es el parámetro que impone la limitación en cuanto a la distancia en CSMA/CD. Debido a la atenuación (debilitamiento de una señal transmitida a medida que se aleja del origen) el mecanismo de detección de colisiones no es apropiado a partir de 2500 metros. Si existen muchos equipos en la red, habrá una gran cantidad de tráfico y se multiplicarán los casos de anulación de colisiones, lo que provocará que es el sistema se haga muy lento.

- *Método de acceso múltiple por detección de portadora con anulación de colisiones (CSMA/CA)*: Este es el método menos popular de los que aquí se describen. En este método, cada equipo indica su intención de transmitir antes de

transmitir los datos. De esta forma, los equipos detectan cuándo puede ocurrir una colisión; con lo cual se pueden evitar. Pero el gran inconveniente que tiene este método es que al tener que informar de la intención de transmitir datos aumenta el tráfico en el cable y se ralentiza el rendimiento de la red.

- *Métodos de paso de testigo*: En este método circula por el cable del anillo, equipo en equipo, un paquete especial denominado *testigo*. Cuando un determinado equipo del anillo necesita enviar datos a través de la red, tiene que esperar a un testigo libre. Cuando se detecta un testigo libre, el terminal se apodera de él si tiene datos que enviar. Mientras un equipo está utilizando el testigo, los demás no pueden transmitir datos. Debido a que sólo puede haber un equipo utilizando el testigo, no puede haber colisiones ni contención y no se pierde tiempo esperando a que los equipos vuelvan a enviar los testigos debido al tráfico de la red.

- *Métodos de prioridad de demandas*: Este método es relativamente nuevo y está diseñado para el estándar Ethernet 100 Mbps, conocido como 100VG-AnyLAN. Este método está basado en el hecho de que los nodos repetidores y finales son los dos componentes que forman todas las redes del tipo citado. Los repetidores gestionan el acceso a la red haciendo búsquedas *Round-Robin* de peticiones de envío de todos los nodos de la red. El repetidor o *Hub* es el responsable de conocer todas las direcciones, enlaces y nodos finales, además de comprobar que todos ellos están funcionando. Al igual que en métodos estudiados anteriormente, si dos equipos tratan de acceder al medio al mismo tiempo, se produce contención. Pero con este esquema es posible implementar que ciertos datos tengan prioridad sobre otros, enviándose primero aquellos cuya prioridad sea más alta. Si existen dos peticiones con la misma prioridad, ambas peticiones se servirán alternando entre las dos. Este método es más eficiente que los anteriores si tenemos en cuanta que en cada momento sólo hay comunicación entre el equipo que envía, el hub y el equipo que recibe. Es por ello por lo que cada hub debe conocer sólo los nodos finales y los repetidores que están conectados a él mientras que CSMA/CD cada hub conoce la dirección de cada nodo de la red.

## 3.10 ENVÍO DE DATOS EN UNA RED

Inicialmente se puede pensar que los datos se envían desde un equipo a otro como una serie continua de unos y ceros. Pero lo que ocurre en realidad es que la información se divide en paquetes pequeños y manejables, cada uno con la información esencial para ir desde el origen hasta el destino.

Normalmente la información se almacena en archivos de gran tamaño. En cambio las redes no podrían funcionar si los equipos colocasen a la vez grandes cantidades de datos en la red. Si esto fuera así todos los equipos tendrían que esperar mientras un usuario trata de enviar gran cantidad de información con lo cual se pasaría de compartir la red a monopolizar la red. Existen dos razones fundamentales por las que la transmisión de grandes paquetes de información es ineficiente:

- Las grandes cantidades de datos enviadas como un único bloque colapsan la red y hacen imposible la interacción y la comunicación apropiada debido a que un equipo está desbordando el cable con datos.
- El impacto de la retransmisión de grandes bloques de datos multiplica el tráfico en la red.

Y como ya hemos explicado, estos efectos se minimizan cuando estos grandes bloques de datos se dividen en paquetes más pequeños para una mejor gestión del control de errores en la transmisión. Además de esta forma los usuarios comparten el acceso a la red.

Cuando el sistema operativo de la red del equipo origen divide los datos en paquetes, añade a cada trama una información de control especial que permite:

- El envío de los datos originales en pequeños paquetes.
- La reorganización de los datos en el orden apropiado cuando lleguen a su destino.
- La comprobación de errores una vez que se hayan reorganizado los datos.

Los componentes básicos de uno de estos paquetes son, además de la información que se desea enviar, los siguientes:

- Una dirección de origen que identifica al equipo que realiza el envío.

- Una dirección de destino que identifica al destinatario.
- Instrucciones que indican a los componentes de la red cómo pasar los datos.
- Información que indica al equipo de destino cómo conectar el paquete con el resto de los paquetes para reorganizar el bloque completo de datos.
- Información de comprobación de errores que asegura que los datos lleguen intactos.

## 3.11 ETHERNET

Existen muchos tipos de redes y arquitecturas que podríamos describir, pero en este documento nos vamos a centrar en la arquitectura de la red Ethernet. El motivo fundamental es que en el laboratorio del departamento de Ingeniería de Sistemas y Automática, donde se ha realizado este proyecto, se encuentra implantada esta configuración. Además, Ethernet se ha convertido en el medio de acceso más conocido para equipos de sobremesa y se utiliza en entornos de redes tanto pequeños como grandes. Este es un estándar que no pertenece a ninguna industria y que ha tenido una gran aceptación por los fabricantes de hardware de red. Casi no existen problemas relacionados con la utilización de productos hardware para Ethernet de distintos fabricantes.

### 3.11.1 ESPECIFICACIONES DE ETHERNET

En 1978, la Organización Internacional de Normalización (ISO) creó un conjunto de especificaciones para la conexión de dispositivos diferentes. Este conjunto de estándares se conoce *como modelo de referencia OSI* . La especificación Ethernet realiza las mismas funciones que los niveles físico y de enlace de este modelo, descrito en otro capítulo de este documento. Estas especificaciones afectan a cómo se conecta el hardware y a cómo se intercambia la información.

En la década de los 80, el IEEE publicó el *Proyecto 802*. Este proyecto generó estándares para el diseño y compatibilidad de los componentes hardware que operaban en los niveles físico y de enlace de datos. El estándar que pertenecía a Ethernet es el 802.3 de IEEE.

### 3.11.2 CARACTERÍSTICAS DE ETHERNET

Esta arquitectura presenta como topología tradicional la de Bus Lineal, aunque también podemos configurarla como Bus en Estrella. La transmisión es en banda base y normalmente a 10Mbps, aunque actualmente se están imponiendo las redes que funcionan a 100 Mbps. El método de acceso que implementa es el CSMA/CD y la especificación que la rige es la 802.3. En cuanto a los tipos de cable que se pueden montar tenemos grueso, fino o UTP.

### 3.11.3 FORMATO DE LA TRAMA ETHERNET

Ethernet divide los datos en paquetes en un formato que es diferente al de los paquetes de otras redes: esta arquitectura divide los datos en *tramas*.

Una trama es un paquete de información transmitido como una unidad. Una trama Ethernet puede tener entre 46 y 1500 bytes y cada una de ellas contiene información de control y tienen la misma estructura básica. En esta estructura se distinguen los siguiente campos:

- *Preámbulo*: Indica el principio de la trama.
- *Destino y origen*.
- *Tipo de trama*: Se utiliza para identificar el protocolo de red.
- *Comprobación de redundancia cíclica(CRC)*: Este es un campo de comprobación de errores.

### 3.11.4 LOS ESTÁNDARES IEEE A 10 Mbps

A continuación pasamos a describir las cuatro topologías Ethernet a 10 Mbps:

- *Estándar 10BaseT*: Esta especificación indica que la transmisión es a 10 Mbps en banda base sobre par trenzado, ya sea apantallado (STP) o sin apantallar (UTP de categorías 3, 4 ó 5). Los conectores que se emplean son los RJ-45 al final del cable. Cada equipo conectado a este tipo de red necesitará un transceiver. Los backbones que se usan para los hubs son de cable coaxial o fibra óptica. La mayoría de las redes de este tipo están configuradas en forma de estrella, pero internamente utilizan un sistema de comunicación en bus como el

de otras configuraciones Ethernet. La longitud máxima de un segmento 10BaseT es de 100 metros. Se pueden utilizar repetidores para aumentar esta limitación. La longitud mínima del cable entre dos equipos es de 2,5 metros. Una LAN de este tipo puede gestionar hasta 1024 equipos.

- *Estándar 10Base2*: Se da este nombre a la especificación 802.3 de IEEE porque transmite a 10 Mbps en un hilo en banda base y puede llevar la señal hasta 185 metros. Este tipo de red utiliza un cable coaxial fino, que tiene un segmento de red máximo de 185 metros y una longitud mínima de 0,5 metros entre estaciones. También existe la limitación de hasta 30 equipos por segmento de 185 metros. El cableado con cable fino utiliza normalmente una topología de bus local. Se define para este estándar la conocida como regla 5-4-3 en la que con un cable fino se combinan hasta cinco segmentos de cable conectados por cuatro repetidores pero en los que sólo puede haber tres segmentos con estaciones conectadas.

- *Estándar 10Base5*: Las especificaciones de IEEE para esta topología son de 10 Mbps y segmentos de 500 metros. También es conocida como Ethernet estándar. Esta topología hace uso de cable coaxial grueso. Normalmente este tipo de cable se monta con una topología en bus y puede soportar hasta 100 nodos (estaciones, repetidores y demás) por segmento backbone. Como ahora un segmento de cable grueso puede tener hasta 500 metros la longitud máxima de la red es de 2500 metros. También se define una regla 5-4-3 para cable grueso y se trata de una configuración igual a la descrita para cable fino pero lo que cambian son las longitudes máximas tal y como se han descrito.

- *Estándar 10BaseFL*: Aquí están contempladas las especificaciones para Ethernet en cable de fibra óptica. La razón para utilizar este tipo de cable es trabajar con segmentos largos entre repetidores, como puede ser entre edificios. La longitud máxima de un segmento para este caso es de 2000 metros.

### 3.11.5 LOS ESTÁNDARES IEEE A 100 Mbps

Los nuevos estándares están abandonando el límite habitual de los 10 Mbps de la Ethernet original. Se están desarrollando estas nuevas posibilidades para atender a aplicaciones que requieren un ancho de banda elevado como pueden ser:

- Diseño asistido por ordenador (CAD).
- Fabricación asistida por ordenador (CAM).
- Vídeo
- Almacenamiento y transmisión de imágenes y documentos.

Dos estándares que se ajustan a estas nuevas demandas son:

- Ethernet 100BaseVG-AnyLAN.
- Ethernet 100BaseX (Fast Ethernet).

Ambos estándares son entre 5 y 10 veces más rápidos que los Ethernet estándar. Además, son bastante compatibles con el cableado de 10BaseT. Esto significa permitir actualizaciones *plug and play* a instalaciones 10BaseT existentes.

El caso de la VG incluye mejoras como:

- Una tasa mínima de 100Mbps.
- La posibilidad de soportar tecnologías en estrella en cascada con cables de par trenzado de Categoría 3, 4 ó 5 y con fibra óptica.
- El método de acceso por prioridad de demandas que permita dos niveles de prioridad (alta y baja).
- La posibilidad de permitir una opción de filtrado de tramas en hub para aumentar la privacidad de la información que circula por nuestra red.
- Soporte para tramas Ethernet y paquetes Token Ring.

La 100BaseX es una extensión del estándar Ethernet existente. Utiliza cable UTP de categoría 5 y utiliza CSMA/CD en una topología de bus en estrella, donde todos los cables están conectados a un hub.

# CAPÍTULO 4

# INTRODUCCIÓN A LOS PROTOCOLOS DE RED

# 4 INTRODUCCIÓN A LOS PROTOCOLOS DE RED

## 4.1 INTRODUCCIÓN

Existen muchos fabricantes de software y de hardware que proporcionan productos para la conexión de equipos en la red. Fundamentalmente, las redes son un medio de comunicación, de ahí que los fabricantes deban tomar medidas para asegurar que sus productos puedan interactuar con productos de otras compañías. Para dirigir los aspectos concernientes a la estandarización, varias organizaciones independientes han creado especificaciones estándar de diseño para los productos de redes de equipos. Cuando se siguen estos estándares, es posible la comunicación entre productos hardware y software de diversos vendedores.

## 4.2 PROCESOS EN LAS COMUNICACIONES DE RED

El complejo proceso de enviar datos de un equipo a otro a través de una red lo podemos dividir en tareas secuenciales discretas de la siguiente forma:

- Reconocer los datos.
- Dividir los datos en porciones manejables.
- Añadir información a cada porción de datos para determinar la ubicación de los datos y para identificar al receptor.
- Añadir información de temporización y verificación de errores.
- Colocar datos en la red y enviarlos por su ruta.

El software de red trabaja a muchos niveles diferentes dentro de los equipos emisores y receptores. Cada uno de estos niveles es gestionado por uno o más protocolos. Estos protocolos son especificaciones estándar para dar formato a los datos y transferirlos. Es decir, se trata de un conjunto de reglas que hacen posible el intercambio de información entre dos equipos informáticos. De esta forma, cuando los equipos emisores y receptores siguen el mismo protocolo se asegura la comunicación.

## 4.3 EL MODELO DE REFERENCIA OSI

ISO es una organización no gubernamental fundada en 1947 y que tiene como misión la coordinación del desarrollo y aprobación de estándares en el ámbito internacional. Su ámbito de trabajo cubre todas las áreas, incluyendo las redes locales, a excepción de las áreas electrotécnicas que son coordinadas por el IEC (International Electrotechnical Commision).

Cada país únicamente puede estar representado en ISO por una organización; en el caso de España, está representada por AENOR (Asociación Española de Normalización) y en el caso de Estados Unidos, está representada por ANSI (American National Standards Institute).

En 1978, la ISO divulgó un conjunto de especificaciones que describían la arquitectura de red para la interconexión de dispositivos diferentes. El documento original se aplicó a sistemas que eran abiertos entre sí, debido a que todos ellos podían utilizar los mismos protocolos y estándares para intercambiar información. En 1984, la ISO presentó una revisión de este modelo y lo llamó *"modelo de referencia de interconexión de sistemas abiertos (OSI)"*. Esta revisión de 1984 se ha convertido en un estándar internacional y se utiliza como guía para las redes.

El modelo OSI es la guía mejor conocida y más ampliamente utilizada para la visualización de entornos de red. Éste ofrece una descripción del funcionamiento conjunto de hardware y software de red por niveles para posibilitar las comunicaciones. El modelo también ayuda a localizar problemas proporcionando un marco de referencia que describe el supuesto funcionamiento de los componentes.

### 4.3.1 ARQUITECTURA EN NIVELES

La arquitectura del modelo de referencia OSI divide la comunicación en siete niveles o capas. Cada nivel cubre diferentes actividades, equipos o protocolos de red. Además este modelo define cómo se comunica y trabaja cada nivel con los niveles inmediatamente superior e inferior. Por ejemplo, el nivel de sesión se comunica y trabaja con los niveles de presentación y transporte. Cada nivel proporciona algún servicio o acción que prepara los datos para entregarlos a través de la red a otro equipo.


**Figura 4.1. Niveles del modelo OSI**

De esta forma, los niveles inferiores definen el medio físico de la red y las tareas relacionadas. Los niveles superiores definen la forma en que las aplicaciones acceden a los servicios de comunicación. Cuanto más alto es el nivel, más compleja es su tarea.

Los niveles están separados entre sí por fronteras llamadas *interfaces*. Todas las demandas se pasan desde un nivel al nivel siguiente a través de la interfaz. Cada capa se basa en los estándares y actividades de la capa inferior.

### 4.3.2 RELACIONES ENTRE NIVELES

En el modelo OSI, cada nivel proporciona servicio al nivel inmediatamente superior y lo protege de los detalles de implementación de los servicios de los niveles inferiores. Al mismo tiempo, cada capa parece estar en comunicación directa con su capa asociada del otro equipo. Esto proporciona una comunicación lógica, o virtual entre niveles análogos. Para esta comunicación entre niveles análogos de equipos distintos, el software implementa las funciones de red de acuerdo con un conjunto de protocolos.

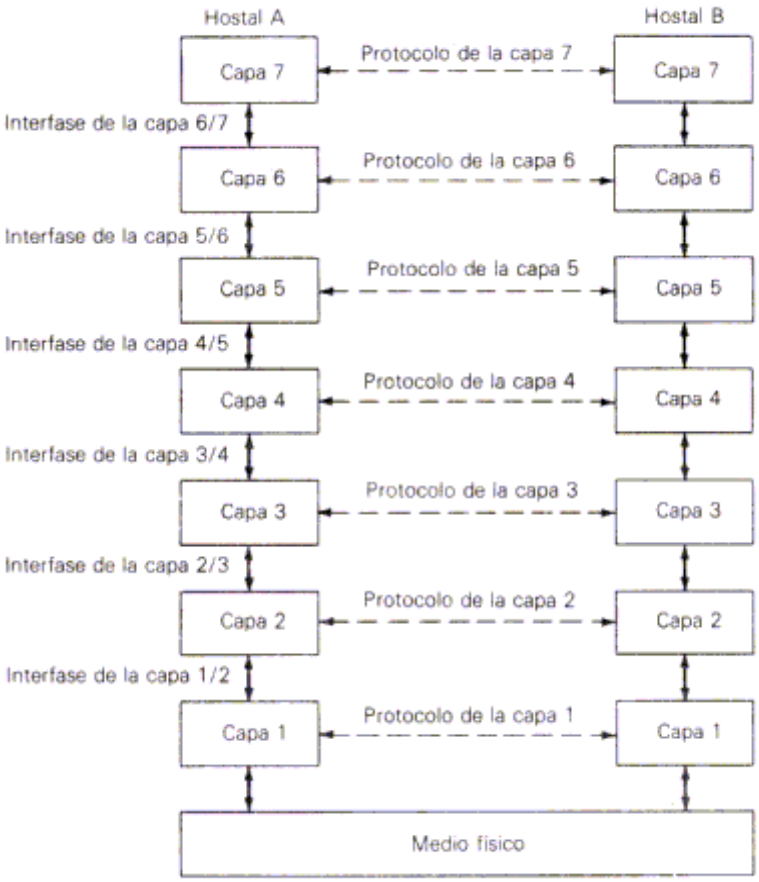



**Figura 4.2. Relaciones del modelo OSI.**

La transmisión de información de un nivel a otro, se realiza dividiendo la misma en paquetes de datos o unidades de información, que se transmiten como un todo desde un dispositivo a otro a través de la red. Esta red pasa un paquete de un nivel software a otro en el mismo orden de los niveles. En cada nivel el software agrega información de formato o de direccionamiento al paquete, que es necesaria para la correcta transmisión del mismo a través de la red.

En el extremo receptor, el paquete pasa a través de las capas en orden inverso. Una utilidad software en cada nivel lee información del paquete, la elimina y pasa el paquete hacia el siguiente nivel superior. Cuando el paquete alcanza la capa de aplicación, la información de direccionamiento ha sido eliminada y el paquete se encuentra en su formato original, con lo que es legible para el receptor. Podemos ver este proceso gráficamente en la figura 4.3.

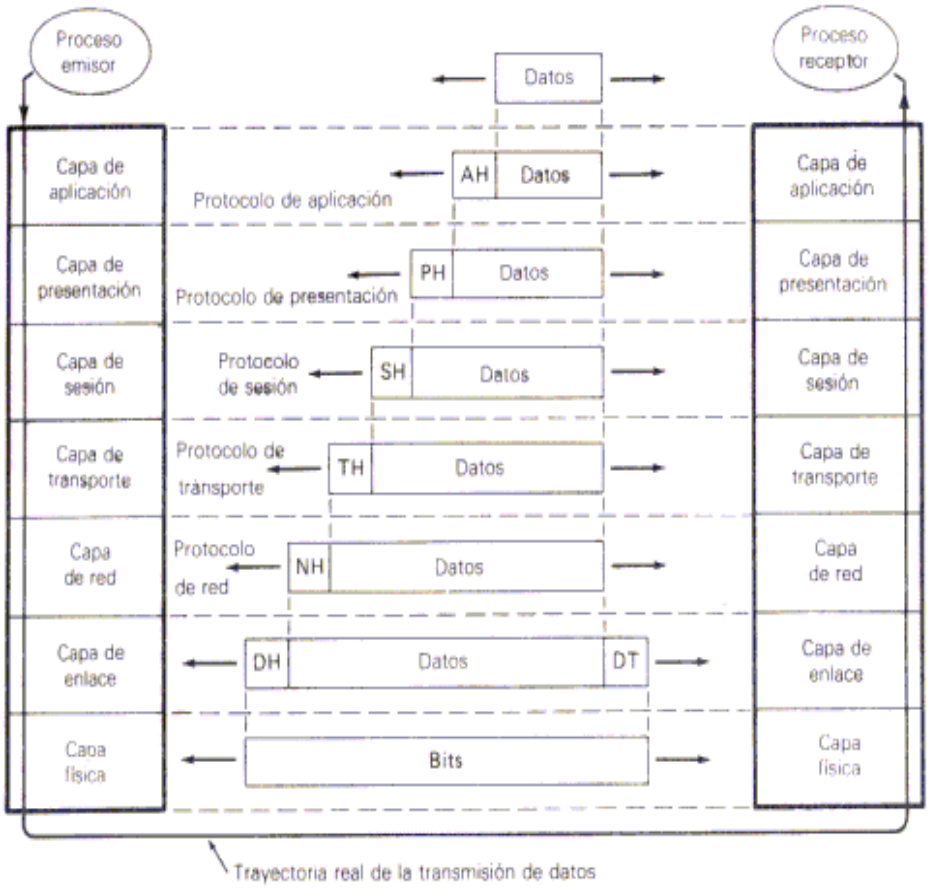



**Figura 4.3. Tratamiento de la información a través de los niveles.**

Con la excepción del nivel más bajo del modelo OSI (el nivel físico), ningún nivel puede pasar datos directamente a su homólogo en el otro equipo. En su lugar, la información del equipo emisor debe ir descendiendo por todos los niveles hasta alcanzar el nivel físico. En ese momento la información se desplaza a través del cable de red hacia el equipo receptor y asciende por sus niveles hasta que alcanza la capa correspondiente. Por ejemplo, cuando el nivel de red envía información acerca del equipo A, la información desciende hacia los niveles de enlace de datos y físico de la parte emisora, atraviesa el cable y asciende los niveles físico y enlace de datos de la parte receptora hasta su destino final en el nivel de red del equipo B.

Los niveles adyacentes dentro de una misma máquina interactúan a través del interfaz. La interfaz define los servicios ofrecidos por el nivel inferior para el nivel superior y además define cómo se accede a dichos servicios.

A continuación pasamos a describir el propósito de cada uno de los siete niveles del modelo OSI e identificar los servicios que cada uno proporciona a sus adyacentes.

### 4.3.3 EL NIVEL DE APLICACIÓN

El nivel 7, el más alto del modelo OSI, es el *nivel de aplicación*. Este nivel se relaciona con los servicios que soportan directamente las aplicaciones de usuario, como software para trasferencia de archivos, acceso a bases de datos, correo electrónico, etc. Dicho de otra forma, esta capa sirve como ventana a través de la cual los procesos de las aplicaciones pueden acceder a los servicios de la red. El envío de un mensaje a través de la red entra al modelo OSI por este punto y sale por el nivel de aplicación del modelo OSI de la máquina destino. Los protocolos de este nivel pueden ser programas en sí mismos, como el FTP *(File Transfer Protocol*) u otros programas relacionados. Las capas inferiores soportan las tareas que se realizan en el nivel de aplicación. Estas tareas incluyen el acceso general a la red, el control de flujo y la recuperación de errores.

### 4.3.4 EL NIVEL DE PRESENTACIÓN

El nivel 6, la *capa de presentación*, define el formato utilizado para el intercambio de información entre dos equipos conectados a la red. Se puede ver como el traductor de la red. Cuando los equipos de diferentes sistemas (como IBM, Apple o Sun) necesitan comunicarse se debe realizar una cierta traducción y reordenación de bytes. Dentro del equipo emisor, el nivel de presentación traduce los datos del formato enviado por el nivel de aplicación en un formato intermedio, generalmente reconocido. En el equipo receptor, este nivel traduce el formato intermedio en un formato que pueda ser útil para el nivel de aplicación de ese equipo. Esta capa de presentación es la responsable de la conversión de protocolos, la traducción de los datos, la encriptación de la información, la modificación o conversión del conjunto de caracteres y la expansión de los comandos básicos. Este nivel también gestiona la compresión de los datos para reducir el número de bits que se necesitan para transmitir, además del control de las impresoras, emulación de terminal y los sistemas de codificación.

### 4.3.5 EL NIVEL DE SESIÓN

El nivel 5 o *nivel de sesión* permite que dos aplicaciones en diferentes equipos abran, utilicen y cierren una conexión llamada *sesión*. Una sesión en un diálogo altamente estructurado entre dos estaciones. Esta capa es la responsable de la gestión de este diálogo. Esta realiza el reconocimiento de nombres y otras funciones, como la seguridad que se necesita para permitir que dos aplicaciones se comuniquen a través de una red.

El nivel de sesión sincroniza las tareas de usuario colocando puntos de control en el flujo de datos. Los puntos de control dividen los datos en grupos más pequeños para la detección de errores. De esta manera, si la red falla, sólo tienen que retransmitirse los datos posteriores al último punto de control. Esta capa también implementa control de diálogo entre los procesos de comunicación, como la regulación de qué parte transmite, cuándo y durante cuánto tiempo.

### 4.3.6 EL NIVEL DE TRANSPORTE

El nivel 4, el *nivel de transporte*, proporciona un nivel de conexión adicional por debajo del nivel de sesión. Esta capa garantiza que los paquetes de datos se envíen sin errores, en secuencia y sin pérdidas o duplicados. En el equipo emisor, este nivel vuelve a empaquetar los mensajes, dividiendo los mensajes grandes en varios paquetes pequeños y agrupando los paquetes pequeños en uno solo. Este proceso asegura que estos bloques se transmitan de forma eficiente a través de la red. En el equipo receptor, el nivel de transporte abre el paquete, reagrupa los mensajes originales y normalmente envía una confirmación de que se recibió el mensaje. Si llega un paquete duplicado, este nivel reconocerá el duplicado y lo descartará.

La capa de transporte proporciona control de flujo y manejo de errores y además participa en la resolución de problemas relacionados con la transmisión y recepción de paquetes. *Transmision Control Protocol* (TCP) *y Sequenced Packet Exchange* (SPX) son ejemplos de protocolos del nivel de transporte.

### 4.3.7 EL NIVEL DE RED

El nivel 3, el *nivel de red*, es el responsable del direccionamiento de los mensajes y la traducción de las direcciones y nombres lógicos en direcciones físicas. Este nivel también determina la ruta desde el equipo origen hasta el equipo destino.

Determina el camino que deben tomar los datos en base a las condiciones de la red, la prioridad de los servicios y otros factores. También gestiona los problemas de tráfico en la red, como la conmutación y el encaminamiento de paquetes y el control de congestión de los datos.

Si la tarjeta de red del router no puede transmitir una porción de datos tan grande como para ser enviada por el equipo origen, el nivel de red del router lo compensa dividiendo los datos en unidades más pequeñas. En el extremo destino, el nivel de red reagrupa los datos. *Internet Protocol* (IP) e *Internet Packet Exchange* (IPX) son ejemplos del protocolo de nivel de red.

### 4.3.8 EL NIVEL DE ENLACE DE DATOS

El nivel dos es el conocido como *nivel de enlace de datos*, y se encarga de enviar tramas de datos desde el nivel de red hacia el nivel físico. Controla los impulsos eléctricos que entran y salen del cable de red. En el extremo receptor, el nivel de enlace de datos empaqueta los bits puros del nivel físico en tramas de datos (una trama de datos es una estructura lógica y organizada en la que se pueden colocar los datos). La representación eléctrica de los datos (patrones de bits, métodos de codificación y testigos) sólo se conocen en este nivel.

Además el nivel de enlace de datos es el responsable de proporcionar una transferencia libre de errores de estas tramas desde un equipo hacia otro a través del nivel físico. Esto permite que el nivel de red prevea una transmisión virtualmente libre de errores sobre la conexión de red.

Normalmente, cuando el nivel de enlace de datos envía una trama, espera una confirmación del receptor. La capa de enlace del receptor detecta cualquier problema que pudiera haber ocurrido con la trama durante la transmisión. Las tramas que se han dañado durante la transmisión o las que no han recibido confirmación se volverán a enviar.

Las normas Ethernet y Token Ring están definidas en este nivel y en el siguiente.

### 4.3.9 EL NIVEL FÍSICO

El nivel 1, el más bajo del modelo OSI, es el nivel físico. Este nivel transmite el flujo de bits puros no estructurados sobre un medio físico (como el cable de red). El nivel físico está completamente orientado al hardware y se encarga de todos los

aspectos de establecimiento y mantenimiento de un enlace físico entre dos equipos a comunicar. El nivel físico también porta las señales que transmiten los datos generados por cada uno de los niveles superiores.

Este nivel define la forma de conectar el cable a la tarjeta de red. Por ejemplo, de cuántas patillas tiene que ser el conector y la función de cada una. También define la técnica de transmisión que se utilizará para enviar los datos sobre el cable de red.

Esta capa proporciona codificación de datos y sincronización de bits. El nivel físico es el responsable de la transmisión de bits (ceros y unos) de un equipo a otro, asegurando que cuando se transmite un bit uno se recibe un uno y no un cero. Puesto que diferentes tipos de medios físicamente transmiten los bits (señales eléctricas o luminosas) de forma distinta, el nivel físico también define la duración de cada impulso y cómo se traduce cada bit en el impulso óptico o eléctrico apropiado por el cable de red.

Es frecuente que se llame a este nivel *nivel hardware.* Aunque el resto de los niveles se pueden implementar como firmware, más que como software real, los otros niveles son software en relación con este primer nivel.

Los sistemas de redes locales más habituales definidos en este nivel son: Ethernet, red en anillo con paso de testigo (*Token Ring*) e interfaz de datos distribuidos sobre fibra óptica (FDI, *Fiber Distributed Data Interface*).

## 4.4 PROTOCOLOS

A lo largo de todo este capítulo hemos hablado de los protocolos. Ha llegado la hora de centrarnos en sus funciones y estructura básica para poder así entender los fundamentos de los protocolos de red que es el objetivo de esta parte del proyecto fin de carrera.

Los protocolos son reglas y procedimientos para la comunicación. El término protocolo se utiliza en distintos contextos, pero a nosotros nos interesa la definición que concierne a la informática. Cuando dos equipos están conectados en red, las reglas y procedimientos técnicos que dictan su comunicación e interacción de denominan protocolos.

Los protocolos de red tienen en común las siguientes características:

- Hay muchos protocolos. A pesar de que cada protocolo facilita la comunicación básica, cada uno tiene un propósito diferente y realiza tareas distintas. Cada protocolo tiene sus propias ventajas y limitaciones.

- Algunos protocolos sólo trabajan en ciertos niveles OSI. El nivel al que trabaja un protocolo describe su función. Por ejemplo, un protocolo que trabaje en el nivel físico, asegura que los paquetes de datos pasen a la tarjeta de red y salgan al cable de red.

- Los protocolos también pueden trabajar juntos en una jerarquía o conjunto de protocolos. Al igual que una red incorpora funciones a cada uno de los niveles del modelo OSI, distintos protocolos también trabajan juntos a distintos niveles en la jerarquía de protocolos que corresponden con el modelo OSI. Por ejemplo, el nivel de aplicación del protocolo TCP/IP se corresponde con el nivel de presentación del modelo OSI. Vistos conjuntamente los protocolos definen una jerarquía de funciones y prestaciones.

### 4.4.1 FUNCIONAMIENTO DE LOS PROTOCOLOS

La operación técnica en que los datos son transmitidos a través de la red se puede dividir en dos pasos discretos, sistemáticos. A cada paso se realizan ciertas acciones que no se pueden realizar en otro paso. Cada paso incluye sus propias reglas y procedimientos, o protocolo.

Los pasos del protocolo se tienen que llevar a cabo en un orden apropiado y que sea el mismo en cada uno de los equipos de la red. En el equipo origen, estos pasos se tienen que llevar a cabo de arriba hacia abajo. En el equipo destino de ha de realizar de abajo hacia arriba.

- En el equipo origen, los protocolos se dividen en secciones más pequeñas, denominadas paquetes, que puede manipular el protocolo. Tras esto se añade a los paquetes información sobre la dirección, de forma que el equipo destino pueda determinar si los datos le pertenecen. Finalmente se preparan los datos para la transmisión a través de la tarjeta de red y enviarlos por el cable de red.

- Por otra parte, en el equipo destino, los protocolos constan de la misma secuencia de pasos pero en orden inverso. En primer lugar se toman los paquetes de datos del cable. Se introducen los paquetes en el equipo a través de la tarjeta

de red. A continuación se extrae de los paquetes de datos toda la información transmitida, eliminando la información añadida por el equipo origen. Tras esto se copian los datos de los paquetes en un buffer para reorganizarlos y finalmente se pasan los datos ya reorganizados a la aplicación de una forma utilizable.

### 4.4 2 PROTOCOLOS EN UNA ARQUITECTURA MULTINIVEL

En una red tienen que trabajar juntos varios protocolos. Al trabajar juntos, aseguran que los datos se preparen correctamente, se transfieran al destino correspondiente y se reciban de forma apropiada. Es por ello por lo que el trabajo de los distintos protocolos tiene que estar coordinado de forma que no se produzcan conflictos o se realicen tareas incompletas. Los resultados de esta coordinación se conoce como *trabajo en niveles*.

Se define una *jerarquía de protocolos* como una combinación de protocolos. Cada nivel de la jerarquía especifica un protocolo diferente para la gestión de una función o de un subsistema del proceso de comunicación. Cada nivel tiene su propio conjunto de reglas. Citemos como ejemplo el modelo OSI y las reglas asociadas a cada nivel.

Se conoce como *proceso de ligadura* al proceso con el que se conectan los protocolos entre sí y con la tarjeta de red. Este permite una gran flexibilidad a la hora de configurar una red. Se pueden mezclar y combinar los protocolos y las tarjetas de red según las necesidades. Por ejemplo, se pueden ligar dos jerarquías de protocolos a una tarjeta de red, como Intercambio de paquetes entre redes e Intercambio de paquetes en secuencia (IPX/SPX) y el Protocolo de control de transmisión/Protocolo Internet (TCP/IP). Si hay más de una tarjeta de red en un equipo, cada jerarquía de protocolos puede estar en una tarjeta o en ambas.

El orden de ligadura determina la secuencia en la que el sistema operativo ejecuta el protocolo. Cuando se ligan varios protocolos a una tarjeta de red, el orden de ligadura es la secuencia en que se utilizarán los protocolos para intentar una comunicación correcta. Por ejemplo, si el primer protocolo ligado es el TCP/IP, el sistema operativo de red intentará la conexión con TCP/IP antes de utilizar otro protocolo. Si falla esta conexión, el equipo tratará de realizar una conexión utilizando el siguiente protocolo en el orden de ligadura.

La industria informática ha diseñado varios tipos de protocolos como modelos estándar. Los fabricantes de hardware y software pueden desarrollar sus productos para ajustarse a cada una de las combinaciones de estos protocolos. Los modelos más importantes incluyen:

- La familia de protocolos ISO/OSI
- La arquitectura de sistemas en red de IBM (SNA).
- Digital DECnet.
- Novell Netware.
- AppleTalk de Apple.
- El conjunto de protocolos de Internet TCP/IP.

Los protocolos existen en cada nivel de estas jerarquías, realizando las tareas especificadas por el nivel. Sin embargo, las tareas de comunicación que tienen que realizar las redes se agrupan en un tipo de protocolo entre tres. Cada tipo está compuesto por uno o más niveles del modelo OSI. Estos tipos son los siguientes:

- *Protocolos de aplicación*: Estos trabajan en el nivel superior del modelo de referencia OSI. Proporcionan interacción entre aplicaciones e intercambio de datos. Entre los más conocidos citaremos: APPC, FTAM, X.400, X.500, FTP, SMTP, SNMP...
- *Protocolos de transporte*: Estos protocolos facilitan las sesiones de comunicación entre equipos y aseguran que los datos se puedan mover con seguridad entre los terminales. Como los más conocidos cabe destacar: TCP, SPX, NWLink, NetBEUI, ATP...
- *Protocolos de red*: Estos proporcionan lo que se denominan “servicios de enlace”. Estos protocolos gestionan información sobre direccionamiento y encaminamiento, comprobación de errores y peticiones de retransmisión. Los protocolos de red también definen reglas para la comunicación en un entorno de red particular como es Ethernet o Token Ring. Los más conocidos son: IP, IPX, NWLink, NetBEUI y DDP.

Una vez hecha esta descripción y a modo de ejemplo podemos decir que la norma de IEEE 802-3 (Ethernet) que estudiamos en el capítulo anterior es un protocolo de comunicación a nivel físico.

## 4.5 TCP/IP

De los protocolos existentes vamos a hacer especial hincapié en el conocido como Protocolo de Control de Transmisión / Protocolo Internet TCP/IP. Nos centraremos en su estudio debido a su popularidad y a que este proyecto fin de carrera se basa en él.

TCP/IP es un conjunto de protocolos aceptados por la industria que permiten la comunicación en un entorno heterogéneo (formado por elementos diferentes). Está diseñado para enlazar ordenadores de diferentes tipos, incluyendo PC's, minis y mainframes, que ejecuten sistemas operativos distintos, sobre redes de área local y redes de área extensa. Además proporciona un protocolo de red encaminable y permite acceder a Internet y a sus recursos. Debido a su popularidad se ha convertido en el estándar de hecho en lo que se conoce como *interconexión de redes*, es decir, la intercomunicación en una red que está formada por redes más pequeñas.

### 4.5.1 INTRODUCCIÓN A TCP/IP

TCP/IP fue desarrollado en 1972 por el Departamento de Defensa de los Estados Unidos, ejecutándose en ARPANET (una red de área extensa del Departamento de Defensa). Su propósito era el de mantener enlaces de comunicación entre sitios en el caso de guerra nuclear. Posteriormente, una red dedicada exclusivamente a aspectos militares denominada MILMET se separó de ARPANET. Esta fue el germen de lo que después constituiría Internet.

TCP/IP se ha convertido en el protocolo estándar para la interoperabilidad entre distintos tipos de equipos. Está diseñado para ser encaminable, robusto y funcionalmente eficiente. Las principales ventajas que ofrece este conjunto de protocolos son:

- Es un estándar en la industria. Por este motivo se trata de un protocolo abierto, es decir, que no está controlado por una única compañía y está sujeto a cuestiones de compatibilidad.

- Contiene un conjunto de utilidades para la conexión de sistemas operativos diferentes.

- Utiliza una arquitectura escalable cliente-servidor. Esto quiere decir que TCP/IP puede ampliarse para ajustarse a las necesidades y circunstancias futuras. Utiliza *sockets* para hacer que el sistema operativo sea algo transparente. Los sockets se describen más adelante en este capítulo ya que son una parte importante de este proyecto fin de carrera.

Históricamente, TCP/IP ha tenido dos grandes inconvenientes: su tamaño y su velocidad.

### 4.5.2 ESTÁNDARES TCP/IP

Los estándares de TCP/IP se publican en una serie de documentos denominados *Request For Comment* (RFC); solicitudes de comentario. Su objetivo principal es proporcionar información o describir el estado de desarrollo. Aunque no se crearon para servir de estándar, muchas RFC has sido aceptadas como estándares.

El desarrollo de Internet está basado en el concepto de estándares abiertos. Es decir, cualquiera que lo desee, puede utilizar o participar en el desarrollo de estándares para Internet. La Plataforma de Arquitectura Internet (IAB) es el comité responsable para la gestión y publicación de las RFC. La IAB permite a cualquier persona o a cualquier compañía que envíe o evalúe una RFC.

### 4.5.3 TCP/IP Y EL MODELO OSI

El protocolo TCP/IP no se corresponde exactamente con el modelo OSI. En vez de tener siete niveles, sólo utiliza cuatro. Normalmente conocido como Conjunto de Protocolos de Internet, TCP/IP se divide en estos cuatro niveles:

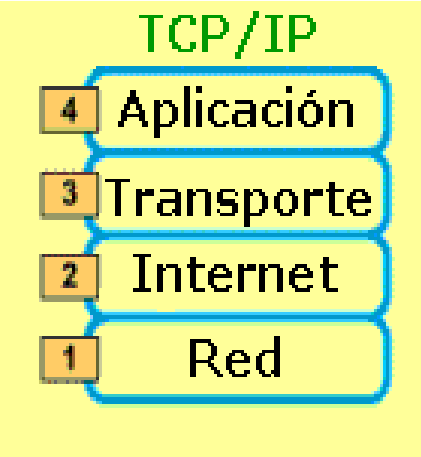



**Figura 4.4. Conjunto de protocolos de Internet.**

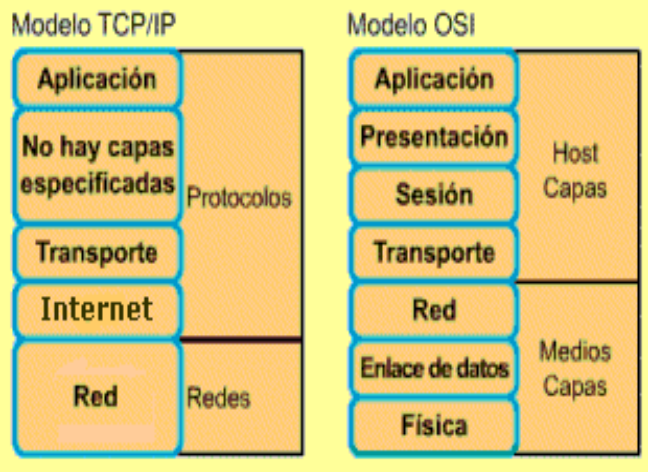



**Figura 4.5. Relación TCP/IP y modelo OSI.**

- **Nivel de interfaz de red**. Se corresponde con los niveles físico y de enlace de datos del modelo OSI y se comunica directamente con la red. Proporciona la interfaz entre la arquitectura de red (como Ethernet) y el nivel de Internet.

- **Nivel Internet**. Se corresponde con el nivel de red del modelo OSI. Utiliza varios protocolos para encaminar y entregar los paquetes. Los elementos de conexión llamados routers, son dependientes del protocolo. Los distintos protocolos que trabajan en el nivel de red son:

  - *Protocolo Internet (IP)*: Es un protocolo de conmutación de paquetes que realiza direccionamiento y encaminamiento. Cuando se transmite un paquete, este protocolo añade una cabecera al paquete, de forma que pueda enviarse a través de la red utilizando las tablas de encaminamiento dinámico.

IP es un protocolo no orientado a conexión y envía paquetes sin esperar la confirmación por parte del receptor (servicio no confirmado). Además este protocolo es el responsable del empaquetado y división de los paquetes requerido por los niveles físico y enlace de datos del modelo OSI. Cada paquete IP está constituido por una dirección de origen y una de destino, un identificador de protocolo, un checksum (un valor calculado) y un TTL (Time To Live). El TTL indica a cada uno de los routers de la red entre el origen y el destino cuánto tiempo le queda el paquete por estar en dicha red. Funciona como un controlador o reloj de cuenta atrás. Cuando el paquete pasa por el router, este reduce el valor en una unidad (un segundo) o el tiempo que llevaba esperando para ser entregado. Cuando la cuenta TTL llega a cero, el paquete es retirado de la red. El propósito de este TTL es evitar que los paquetes perdidos o dañados estén circulando indefinidamente.

o *Protocolo de Resolución de Direcciones (ARP)*: Antes de enviar un paquete a otro host, se tiene que conocer la dirección hardware de la maquina receptora. El ARP determina la dirección hardware que corresponde a una dirección IP. Si ARP no contiene la dirección en su propia caché, envía una petición por toda la red solicitando la dirección. Todos los host de la red procesan la petición y si contienen un valor para esa dirección lo devuelven al solicitante. A continuación se envía el paquete a su destino y se guarda la información de la nueva dirección en la caché del router.

o *Protocolo Inverso de Resolución de Direcciones(RARP)*: Un servidor RARP mantiene una base de datos de números de máquina en la forma de una tabla (o caché) ARP que está creada por el administrador del sistema. A diferencia del caso anterior, este protocolo proporciona una dirección IP a una petición con dirección de hardware. Cuando el servidor RARP recibe una petición de un número IP desde un nodo de la red, responde comprobando su tabla de encaminamiento para el número de máquina del nodo que realiza la petición y devuelve la dirección IP al nodo que realizó la petición.

o *Protocolo de Mensajes de Control de Internet (ICMP)*: Este protocolo es utilizado por IP y superiores para enviar y recibir informes de estado sobre la

información que se está transmitiendo. Los routers suelen usar ICMP para controlar el flujo de datos entre ellos.

- **Nivel de transporte**. Se corresponde con el nivel de transporte del modelo OSI. Es el responsable de establecer y mantener una comunicación entre dos hosts. Este nivel proporciona notificación de la recepción, control de flujo y secuenciación de paquetes. También gestiona las retransmisiones de paquetes. El nivel de transporte puede utilizar los protocolos TCP o el protocolo de datagramas de usuario (UDP) en función de los requerimientos de la transmisión. Veamos cada uno de ellos:

  - *Protocolo de Control de Transmisión (TCP)*: Este es el responsable de la transmisión de datos fiable desde un nodo a otro. Es un protocolo orientado a la conexión y establece una conexión (también conocida como una sesión, circuito virtual o enlace) entre dos máquinas antes de transferir ningún dato. Para establecer una conexión fiable, TCP utiliza lo que se conoce como acuerdo en tres pasos. En primer lugar el solicitante envía al servidor un paquete especificando el número de puerto que él planea utilizar y el número de secuencia inicial (ISN). Tras esto el servidor responde con su ISN, que consiste en el ISN del solicitante más uno. Finalmente, el solicitante responde a la respuesta del servidor con el ISN del servidor más uno. Para mantener una conexión fiable, cada paquete deberá contener un número de puerto TCP origen y destino, un número de secuencia para mensajes que tienen que dividirse en partes más pequeñas, un checksum que asegura que la información se ha transmitido sin error y un número de confirmación que indica a la máquina origen qué partes de la información han llegado.

  - *Protocolo de Datagramas de Usuario (UDP)*: El UDP, un protocolo no orientado a conexión, es el responsable de la comunicación de datos extremo a extremo. Como UDP no establece conexión, cuando envía los datos trata de comprobar que el host de destino los recibe. UDP se utiliza para enviar pequeñas cantidades de datos que no necesitan una entrega garantizada. Aunque UDP utiliza puertos, son distintos de los puertos TCP; así pues se pueden utilizar los mismos números sin interferirse. Sobre puertos y sockets hablaremos más adelante por ser parte importante en este proyecto fin de carrera.

- **Nivel de aplicación**. El nivel de aplicación se corresponde con los niveles de sesión, aplicación y presentación del modelo OSI y conecta las aplicaciones a la red. Dos interfaces de aplicaciones (API) proporcionan acceso a los protocolos de transporte TCP/IP, los sockets de Windows y NetBIOS.

De una forma gráfica se puede apreciar en la figura 4.7.

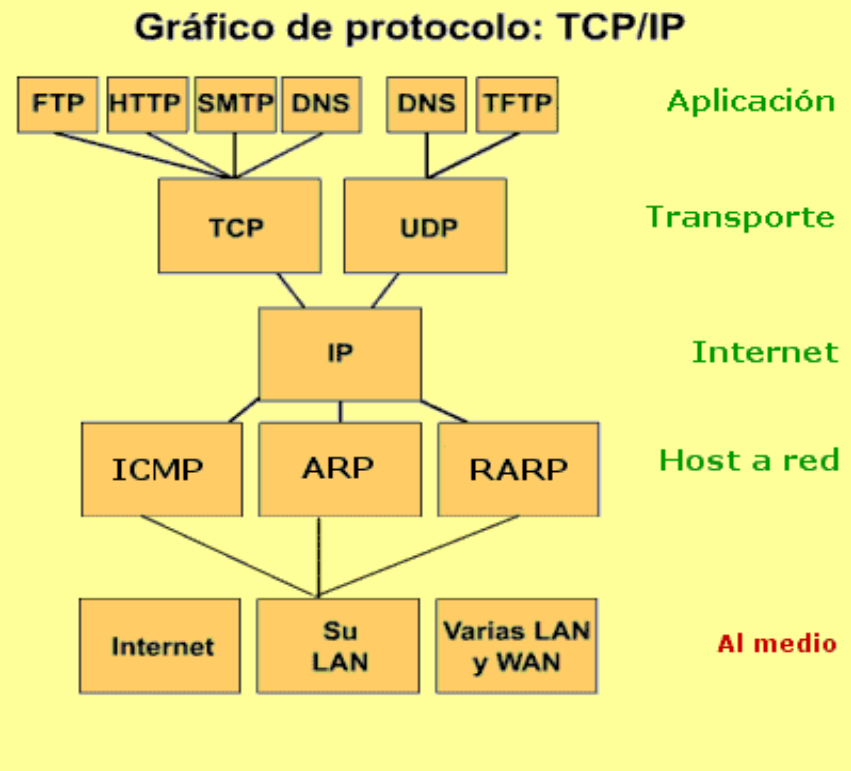



**Figura 4.6. Gráfico de protocolos.**

### 4.5.4 PUERTOS Y SOCKETS

En este proyecto fin de carrera se ha adoptado la solución de comunicar mediante sockets los programas Cliente y Servidor, tal y como hemos descrito en capítulos anteriores de este documento. Pues ahora poseemos los conocimientos básicos necesarios sobre redes y protocolos de red para poder comprender qué es un socket y cómo se utiliza.

La mayoría de los Sistemas Operativos actuales soportan el multiproceso, mediante el cual varios procesos se pueden estar ejecutando a la vez en una máquina. De estos procesos puede haber algunos de ellos comunicándose por red al mismo tiempo, con la misma máquina o con máquinas diferentes. Los números de puerto de protocolo se utilizan para hacer referencia a la localización de una

aplicación o proceso particular en cada máquina (en el nivel de aplicación). En este entorno, decir que un proceso de la máquina receptora es el destino final de un datagrama concreto enviado por otro proceso de la máquina emisora es una afirmación que presta a confusión, ya que los procesos son dinámicos, se crean y se destruyen constantemente, por lo que un proceso en el ordenador emisor en un momento dado no puede saber qué proceso del equipo receptor es el destinatario de los paquetes que está enviando. Además, para una correcta comunicación necesitamos saber qué función del equipo receptor es la encargada de recibir los paquetes, independientemente del proceso que ha lanzado dicha función. Al igual que una dirección IP identifica la dirección la dirección de un host de la red, el número de puerto identifica la aplicación a nivel de transporte, por lo que proporciona una conexión completa de una aplicación de un host a una aplicación de otro host.

Las aplicaciones y servicios pueden configurar hasta 65.536 puertos ya que cada puerto de protocolo viene identificado por un número de 16 bits. Las aplicaciones y servicios TCP/IP suelen utilizar los primeros 1.023 puertos. TCP combina la asignación estática de números de puerto con la asignación dinámica, mediante la denominada asignación de puertos bien conocidos para aplicaciones servidoras (asignación estática) y la asignación del resto de los puertos disponibles a las aplicaciones cliente conforme los vayan necesitando (asignación dinámica). De esta forma cualquier aplicación cliente puede asignar números de puerto dinámicamente cuando sea necesario.

Los números de puerto tienen los siguientes intervalos asignados:

- Los números inferiores a 255 se usan para aplicaciones públicas.
- Los números del 255 al 1023 son asignados a empresas para aplicaciones comercializables.
- Los números superiores a 1023 no están regulados, y se usan generalmente para asignación dinámica.

Para optimizar su funcionamiento los puertos poseen una memoria intermedia, denominada buffer, situadas entre los programas de aplicación y la red. Las aplicaciones transmiten los datos a los puertos, guardándolos éstos en sus buffers hasta que pueden ser enviados por la red. Cuando llegan al host destino, los datos

son almacenados de nuevo en los buffers hasta que la aplicación receptora esté preparada para recibirlos.

Pues tenemos que una dirección IP identifica una máquina de la red y un número de puerto identifica una aplicación dentro de cada máquina. Se define ahora un *socket* como un número de puerto y una dirección IP.

*SOCKET = DIRECCIÓN IP + NÚMERO DE PUERTO*

Actualmente las direcciones IP (Ipv4) poseen 32 bits formados por cuatro campos de 8 bits (octeto), cada uno separado por puntos. La estructura por tanto de una dirección IP es la siguiente:

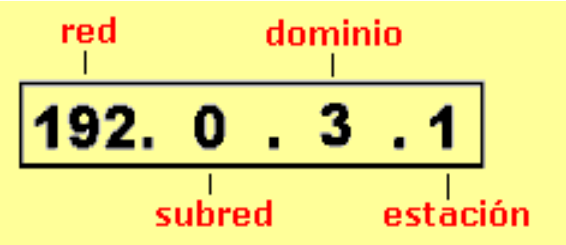



**Figura 4.7. Formato de una dirección IP.**

Los servicios y aplicaciones utilizan sockets para establecer conexiones con otros hosts. Si las aplicaciones necesitan garantizar la entrega de los datos, el socket elige el servicio orientado a conexión (TCP). Si la aplicación no necesita garantizar la entrega de los datos, el socket elige el servicio no orientado a conexión (UDP).

Por tanto, una conexión viene definida por dos puntos extremos (sockets), permitiendo TCP que varias conexiones compartan un punto extremo final (conexiones múltiples a la vez), al estar cada una de ellas identificada por una pareja IP-Puerto de forma única.

Los sockets de Windows (WinSock) son una API de red diseñada para facilitar la comunicación entre aplicaciones y jerarquías de protocolos TCP/IP diferentes. Se definió para que las aplicaciones que utilizasen TCP/IP pudiesen escribir en una interfaz estándar. WinSock se deriva de los sockets originales que creó la API para el sistema operativo UNIX BSD. WinSock proporciona una interfaz común para las

aplicaciones y protocolos que existen cerca de la cima del modelo de referencia TCP/IP. Cualquier programa o aplicación escrita utilizando la API de WinSock, como es el caso de los programas que forman parte de este proyecto fin de carrera, se puede comunicar con cualquier protocolo TCP/IP y viceversa.

Un proceso crea un socket sin nombre cuyo valor de vuelta es un descriptor sobre el que se leerá o escribirá, permitiéndose una comunicación *bidireccional*, característica propia de los sockets y que los diferencia de los *pipes*, o canales de comunicación unidireccional entre procesos de una misma máquina. El mecanismo de comunicación vía sockets tiene los siguientes pasos:

- El proceso servidor crea un socket con nombre y espera la conexión.
- El proceso cliente crea un socket sin nombre.
- El proceso cliente realiza una petición de conexión al socket servidor.
- El cliente realiza la conexión a través de su socket mientras el proceso servidor mantiene el socket servidor original con nombre.

Todo socket viene definido por dos características fundamentales:

- El *tipo del socket*, que indica la naturaleza del mismo, el tipo de comunicación que puede generarse entre los sockets.
- El *dominio del socket* especifica el conjunto de sockets que pueden establecer una comunicación con el mismo.

Para establecer una conexión mediante sockets es necesario tener en cuenta dos características de los mismos: la familia de sockets que se está utilizando, y el tipo de conexión.

La familia de sockets indica la familia o tipo de direcciones que se utilizarán para especificar las diferentes máquinas. Las más comunes son la familia de protocolos internos de Unix, que se utiliza para comunicar procesos que se ejecutan en la misma máquina, y la familia de protocolos Internet, que se utiliza para la transmisión de protocolos como el TCP y el UDP. Existen también otras familias menos frecuentes, como las de la Norma X.25 del CCITT, etc.

El tipo de conexión puede ser de circuito virtual (orientado a conexión) o de tipo datagrama (no orientado a conexión). En el primero se crea una conexión

permanente entre los dos ordenadores a conectar, de forma que una vez realizada la conexión todos los mensajes recorrerán el circuito establecido. En el segundo no se crea una conexión permanente. Los mensajes pueden seguir rutas distintas y no se garantiza que lleguen en el mismo orden en que fueron enviados.

En cuanto a los tipos de sockets que podemos encontrar diremos que estos definen las propiedades de las comunicaciones en las que se ve envuelto un socket, esto es, el tipo de comunicación que se puede dar entre cliente y servidor.

Estas características de la conexión pueden ser:

- Fiabilidad de transmisión.
- Mantenimiento del orden de los datos.
- No duplicación de los datos.
- El "Modo Conectado" en la comunicación.
- Envío de mensajes urgentes.

Los tipos de sockets disponibles son los siguientes:

- *Tipo SOCK_DGRAM*: Son un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió. El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

- *Tipo SOCK_STREAM:* Para comunicaciones fiables en modo conectado, de dos vías y con tamaño variable de los mensajes de datos. Por debajo, en dominios Internet, subyace el protocolo TCP. El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

- *Tipo SOCK_RAW:* Son sockets que dan acceso directo a la capa de software de red subyacente o a protocolos de más bajo nivel. Se utilizan sobre todo para la depuración del código de los protocolos.
- *Tipo SOCK_SEQPACKET:* Tiene las características del SOCK_STREAM pero además el tamaño de los mensajes es fijo.

Por otro lado se encuentra el dominio de un socket, que indica el formato de las direcciones que podrán tomar los sockets y los protocolos que soportarán.

La estructura genérica es:

```
struct sockaddr
{
        u__short sa__family; /* familia */
        char sa__data[14]; /* dirección */
};
```

Pueden ser:

- *Dominio AF_UNIX ( Address Family UNIX* ): El cliente y el servidor deben estar en la misma máquina. Debe incluirse el fichero cabecera /usr/include/sys/un.h. La estructura de una dirección en este dominio es:

```
struct sockaddr__un
{
        short sun__family; /* en este caso AF_UNIX */
        char sun__data[108]; /* dirección */
};
```

- *Dominio AF_INET ( Address Family INET )*: El cliente y el servidor pueden estar en cualquier máquina de la red Internet. Deben incluirse los ficheros cabecera /usr/include/netinet/in.h, /usr/include/arpa/inet.h, /usr/include /netdb.h. La estructura de una dirección en este dominio es:

```
struct in__addr
{
        u__long s__addr;
};
struct sockaddr__in
{
        short sin_family; /* en este caso AF_INET */
        u__short sin_port; /* numero del puerto */
        struct in__addr sin__addr; /* direcc Internet */       char
  sin_zero[8]; /* campo de 8 ceros */
};
```

Estos dominios van a ser los utilizados en xshine. Pero existen otros como:

- *Dominio AF_NS*: Servidor y cliente deben estar en una red XEROX.
- *Dominio AF_CCITT*: Para protocolos CCITT, protocolos X25, ...

Una vez descritos lo anterior, se nos plantea una posibilidad de elección del tipo de socket que debemos utilizar. La decisión depende de la aplicación cliente/servidor que estemos escribiendo. Vamos a ver algunas diferencias entre los protocolos para ayudar en la decisión:

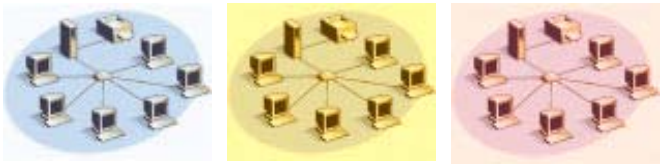
- En UDP, cada vez que se envía un datagrama, hay que enviar también el descriptor del socket local y la dirección del socket que va a recibir el datagrama, luego éstos son más grandes que los TCP. Como el protocolo TCP está orientado a conexión, tenemos que establecer esta conexión entre los dos sockets antes de nada, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no existe en UDP.
- En UDP hay un límite de tamaño de los datagramas, establecido en 64 Kbytes, que se pueden enviar a una localización determinada, mientras que TCP no tiene límite; una vez que se ha establecido la conexión, el par de sockets

funciona como los streams: todos los datos se leen inmediatamente, en el mismo orden en que se van recibiendo.

- UDP es un protocolo *desordenado*, no garantiza que los datagramas que se hayan enviado sean recibidos en el mismo orden por el socket de recepción. Al contrario, TCP es un protocolo *ordenado*, garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado.

- Los datagramas son bloques de información del tipo *lanzar y olvidar*. Para la mayoría de los programas que utilicen la red, el usar un flujo TCP en vez de un datagrama UDP es más sencillo y hay menos posibilidades de tener problemas. Sin embargo, cuando se requiere un rendimiento óptimo, y está justificado el tiempo adicional que supone realizar la verificación de los datos, los datagramas son un mecanismo realmente útil.

En resumen, TCP parece más indicado para la implementación de servicios de red como un control remoto (*rlogin*, *telnet*) y transmisión de ficheros (*ftp*); que necesitan transmitir datos de longitud indefinida. Por ello este ha sido el método elegido para implementar las aplicaciones de este proyecto fin de carrera.



# CAPÍTULO 5

# COMPRESIÓN DE IMÁGENES

# 5 COMPRESIÓN DE IMÁGENES

## 5.1 INTRODUCCIÓN

Tal y como se ha introducido en el capítulo 2 de este proyecto fin de carrera, para poder integrar nuestro trabajo de la mejor forma posible en el entorno donde va a ser utilizado, hemos creído conveniente incluir la posibilidad de comprimir la información antes de transmitirla por la red. A su vez, la información es recuperada por la aplicación Cliente y mostrada al usuario sin ningún tipo de modificación con respecto a su origen. Todo el proceso se realiza de forma completamente trasparente al usuario.

El objetivo principal por el que se realiza la compresión es el de descargar la red en la medida de lo posible y agilizar las tareas de transmisión de la información. Haciendo referencia al capítulo 3 de esta memoria, hemos de recordar que la red de área local instalada en el edificio de laboratorios presenta una topología en bus y posee también una gran cantidad de equipos conectados a ella. Por esta razón, la optimización del tiempo de acceso al medio es fundamental para un funcionamiento adecuado en un entorno tan compartido.

Por tanto, puesto que se ha realizado un estudio exhaustivo de las técnicas de compresión que se utilizan hoy día, se ha creído conveniente incluir un capítulo que muestre los resultados de la investigación y proporcionen al lector el grado de conocimiento necesario para entender las motivaciones que nos han llevado a elegir el algoritmo de Huffman Adaptativo como solución conveniente para cubrir las necesidades de este trabajo.

Asimismo este capítulo, sin ánimo de presunción, podría servir como punto de partida o de referencia para otros trabajos que se interesen por las técnicas de compresión que aquí se describen y que son fruto de un proceso de investigación que facilitaría la tarea a la persona que empiece desde cero a estudiar este tema.

## 5.2 GENERALIDADES DE LA COMPRESIÓN DE IMÁGENES

Cuando se hace un muestreo y se cuantifica una función bidimensional de la intensidad para crear una imagen digital, se produce una enorme cantidad de datos.

De hecho, esta cantidad de datos generados puede ser tan grande que su almacenamiento, procesamiento y sobre todo comunicación pueden llegar a ser desmesurados para cualquier aplicación práctica. En estos casos, es necesario hacer uso de representaciones que vayan más allá del muestreo bidimensional y la cuantificación de niveles de gris.

La compresión de imágenes afronta el problema de la reducción de la cantidad de datos necesarios para representar una imagen digital. La base del proceso de reducción de datos consiste en la eliminación de los datos redundantes. El código de una imagen representa el cuerpo de la información mediante un conjunto de símbolos. La eliminación del código redundante consiste en utilizar el menor número de símbolos para representar la información. Desde el punto de vista matemático, equivale a transformar una distribución bidimensional de píxeles en un conjunto de datos estadísticos sin correlacionar. La transformación se aplica antes del almacenamiento o transmisión de la imagen. Posteriormente, la imagen comprimida se descomprime para reconstruir la imagen original o una aproximación de la misma.

El enfoque que se daba a las primeras investigaciones en este campo de la ingeniería, consistía en el desarrollo de métodos analógicos que permitiesen la reducción del ancho de banda de la transmisión de video, proceso que se denomina *compresión del ancho de banda*. La introducción de los ordenadores digitales y el posterior desarrollo de los circuitos digitales avanzados, hicieron que el interés se desplazara de las técnicas de compresión analógicas a las digitales. Con la reciente adopción de algunos estándares internacionales de compresión de imágenes, este campo está avanzando significativamente gracias a la aplicación práctica de los trabajos teóricos que se iniciaron en 1940 cuando C. E. Shannon y otros colegas desarrollaron por primera vez la visión probabilística de la información, así como de su representación, transmisión y compresión.

Con los años, la necesidad de comprimir las imágenes ha crecido constantemente. En la actualidad se la conoce como una *tecnología de validación*. Por ejemplo, la compresión de imágenes ha sido y continúa siendo crucial para el crecimiento de la información multimedia y el desarrollo de las aplicaciones orientadas a funcionar en la red Internet. Asimismo, es la tecnología natural para solventar los cada vez más específicos planteamientos espaciales de los sensores de imágenes actuales y para la

renovación de los estándares de la señal de televisión. También podemos añadir que la compresión de imágenes desempeña un papel fundamental en diversas aplicaciones de importancia, como las videoconferencias, los sensores remotos, las imágenes de documentos, las imágenes médicas, la transmisión de faxes y control remoto de vehículos no tripulados para aplicaciones militares, espaciales y manipulación de materias peligrosas.

## 5.3 FUNDAMENTOS DE LA COMPRESIÓN DE IMÁGENES

El término *compresión de datos* se refiere al proceso de reducción del volumen de datos necesario para representar una determinada cantidad de información. Debe hacerse una clara distinción entre *datos* e *información*. Los datos son los medios a través de los que se conduce la información. Los datos son una forma de representar la información. De hecho, se pueden utilizar distintas cantidades de datos para describir la misma información. Por tanto, algunas representaciones de la misma información contienen datos redundantes. Esto es lo que se conoce como *redundancia de los datos*.

Esta redundancia de los datos es un punto clave de la compresión de datos digitales. No es un concepto abstracto, sino una entidad matemáticamente cuantificable. Si $n_1$ y $n_2$ representan el número de unidades de información de dos conjuntos de datos que representan la misma información, la *redundancia relativa de los datos* $R_D$ del primer conjunto de datos se puede definir como:

$$R_D = 1 - 1/C_R$$

Donde $C_R$, habitualmente conocido como *Relación de Compresión,* es:

$$C_R = n_1/n_2$$

En el caso en que $n_2 = n_1$, $C_R = 1$ y $R_D = 0$. Esto indica que la primera representación no contiene datos redundantes con respecto a la segunda representación.

En cambio, si $n_2 << n_1$, $C_R \rightarrow \infty$ y $R_D \rightarrow 1$, lo que nos indica que existe una compresión significativa y datos altamente redundantes.

Finalmente, si $n_2 >> n_1$, $C_R \rightarrow 0$ y $R_D \rightarrow -\infty$, que indica que el conjunto de datos de la segunda imagen contiene muchos más datos que la representación original. Desde luego, este caso es habitualmente indeseable de expansión de los datos.

Una relación de compresión práctica, como 10:1 significa que el primer conjunto de datos contiene 10 unidades de transporte de información (bits) por cada unidad del segundo conjunto de datos (comprimidos). La redundancia correspondiente, 0.9, implica que el 90 % de los datos del primer conjunto son redundantes.

En la compresión digital de imágenes, se puede identificar y aprovechar tres tipos básicos de redundancias, la *redundancia de codificación*, *la redundancia entre píxeles* y la *redundancia psicovisual*. La compresión de datos se consigue cuando una o varias de estas redundancias se reducen o eliminan.

### 5.3.1 REDUNDANCIA DE LA CODIFICACIÓN

Es posible obtener una gran cantidad de información sobre la apariencia de una imagen a partir de un histograma de niveles de gris. Un histograma de una imagen es una representación gráfica X-Y, donde en el eje X se representan los posibles valores de niveles de gris que puede tomar un píxel de una imagen y en el eje Y se representa el número de veces que aparece en la imagen un píxel con ese determinado tono de gris.

Una vez dicho esto, podemos explicar cómo el histograma de niveles de gris puede proporcionar una gran cantidad de datos que pueden servir de orientación para la construcción de códigos que reduzcan la cantidad de datos necesarios para representar dicha imagen.

Supongamos que una variable aleatoria discreta $r_k$ del intervalo [0,1] representa niveles de gris de una imagen y que cada $r_k$ sucede con una probabilidad $p_r(r_k)$:

$$p_r(r_k) = n_k / n \qquad k = 0,1,2,...,L-1$$

Donde L es el número de niveles de gris, $n_k$ el número de veces que aparece en la imagen el k-ésimo nivel de gris, y n el número total de píxeles de la imagen. Si el número de bits empleados para representar cada valor de $r_k$ es $l(r_k)$, el promedio de bits necesarios para representar cada píxel es:

$$L_{med} = \Sigma_{k\in[0,L-1]}(l(r_k)p_r(r_k))$$

Así el número total de bits necesarios para codificar una imagen NxM es $MxNxL_{med}$.

Si se representan los niveles de gris de una imagen mediante código binario natural de m bits, se logra reducir el término $L_{med}$ a m bits.

Cuando a cada nivel de gris le asociamos símbolos con la misma longitud en bits estamos realizando la que se conoce como *Codificación de Longitud Fija*. En esta, se asignan palabras de código de longitud iguales a cada símbolo en un alfabeto A sin tener en cuenta sus probabilidades. Si el alfabeto tiene M símbolos diferentes (o bloques de símbolos), entonces la longitud de las palabras de código es el entero más pequeño mayor que $\log_2 M$.

Dos esquemas de codificación de longitud fija comúnmente usados son los códigos naturales y los códigos Gray, que se muestran la figura 5.1 para el caso de una fuente de cuatro símbolos. Nótese que en la codificación Gray, las palabras de código consecutivas difieren en un solo bit. Esta propiedad de los códigos Gray puede proveer una ventaja para la detección de errores. Puede mostrarse que la codificación de longitud fija sólo es óptima cuando:

- El número de símbolos es igual a una potencia de dos
- Todos los símbolos son equiprobables.

Sólo entonces podría la entropía de la fuente ser igual a la longitud promedio de las palabras código que es igual a la longitud de cada palabra código en el caso de la codificación de longitud fija. Para el ejemplo mostrado en la figura 5.1, la entropía de la fuente y la longitud media de la palabra código es 2, asumiendo que todos los símbolos son igualmente probables. A menudo, algunos símbolos son más probables que otros, donde sería más ventajoso usar codificación de la entropía. Realmente, la meta de un sistema de compresión de imágenes es obtener un conjunto de símbolos con una distribución de probabilidad inclinada, para minimizar la entropía de la fuente transformada.

| Símbolo | Código natural | Código Gray |
|---|---|---|
| $a_1$ | 00 | 00 |
| $a_2$ | 01 | 01 |
| $a_3$ | 10 | 11 |
| $a_4$ | 11 | 10 |

**Figura 5.1. Ejemplos de codificación de longitud fija.**

En cambio el método más simple de compresión de imágenes sin pérdidas consiste en reducir únicamente la redundancia de la codificación. Esta redundancia está normalmente presente en cualquier codificación binaria natural de los niveles de gris de una imagen. Dicha redundancia se puede eliminar construyendo un código de longitud variable que asigne las palabras código más pequeñas a los niveles de gris más probables. A este procedimiento se le conoce como *Codificación de Longitud Variable*.

Existen varios métodos de codificación de longitud variable, pero los más usados son la codificación Huffman y la codificación aritmética.

### 5.3.2 REDUNDANCIA ENTRE PÍXELES

La mayoría de las imágenes presentan semejanzas o correlaciones entre sus píxeles. Estas correlaciones se deben a la existencia de estructuras similares en las imágenes, puesto que no son completamente aleatorias. De esta manera, el valor de un píxel puede emplearse para predecir razonablemente el de sus vecinos. En este caso, la información que aporta individualmente un píxel es relativamente pequeña. La mayor parte de la contribución visual de un único píxel a una imagen es redundante; podría haberse inferido de acuerdo con los valores de sus vecinos. En relación con estas dependencias entre píxeles se han acuñado una serie de nombres como *Redundancia Espacial*, *Redundancia Geométrica* o *Redundancia Interna*.

Con el fin de reducir las redundancias entre píxeles de una imagen, la distribución bidimensional normalmente empleada para la percepción e interpretación humana debe ser transformada a un formato más eficaz (aunque habitualmente no visualizable). Por ejemplo, se pueden utilizar las diferencias entre píxeles adyacentes para representar una imagen. Las transformaciones de este tipo (es decir, aquellas

que eliminan la redundancia entre píxeles) se denominan genéricamente *correspondencias*. Se dice que son *reversibles* si los elementos originales de la imagen se pueden reconstruir a partir del conjunto de datos transformados.

Las técnicas de compresión Lempel-Ziv , por ejemplo, implementan algoritmos basados en sustituciones para lograr la eliminación de esta redundancia.

### 5.3.3 REDUNDANCIA PSICOVISUAL

La iluminación de una región, tal y como es percibida por el ojo humano, depende de otros factores además de la luz reflejada por esta región. Por ejemplo, las variaciones de la intensidad (bandas de Mach) pueden ser percibidas en un área de intensidad constante. Este fenómeno se produce porque el ojo humano responde con diferente sensibilidad a la información visual que recibe. Cierta información simplemente tiene menor importancia relativa que otra en el proceso de visión normal. Se dice que esta información es *psicovisualmente redundante* y se puede eliminar sin que se altere significativamente la calidad de la percepción de la imagen.

El que existan redundancias psicovisuales es un hecho natural, ya que la percepción humana de la información de una imagen no consiste en un análisis cuantitativo de cada píxel o de cada valor de luminancia de la imagen. En general, un observador busca características diferenciadoras, como bordes o regiones de diferentes texturas y luego las combina mentalmente en grupos reconocibles. Acto seguido, el cerebro relaciona estos grupos con el conocimiento previo con el fin de completar el proceso de interpretación de la imagen.

Al contrario que la redundancia de codificación y la redundancia entre píxeles, la redundancia psicovisual está asociada a la información real o cuantificable. Su eliminación es únicamente posible porque la propia información no es esencial para el procesamiento visual normal. Como la eliminación de los datos psicovisualmente redundantes se traduce en una pérdida de información cuantitativa, a menudo se denomina con el nombre de *cuantificación*. Puesto que es una operación irreversible (se pierde información visual), la cuantificación conduce a una compresión con pérdida de datos. Técnicas de compresión como JPEG, EZW o SPIHT hacen uso de la cuantificación.

### 5.3.4 CRITERIOS DE FIDELIDAD

Puesto que la eliminación de los datos psicovisualmente redundantes conlleva una pérdida real, o cuantitativa de información visual existe la posibilidad de que la información de interés pueda perderse. Por tanto es conveniente el desarrollo de un método repetible o reproducible que nos permita la cuantificación de la naturaleza y el alcance de la pérdida de información. Sin extendernos en exceso, puesto que la comprensión con pérdidas no está dentro de los propósitos de este proyecto, diremos que se utilizan dos clases de criterios para la valoración de estas pérdidas:

- *Criterios de fidelidad objetiva.*
- *Criterios de fidelidad subjetiva.*

Cuando se puede expresar el nivel de pérdida de información como una función de la imagen original, o de entrada, y de la imagen de salida comprimida y que posteriormente se descomprime, entonces se dice que nos basamos en un *criterio de fidelidad objetiva*. Como ejemplo de este tipo de criterio podemos citar el cálculo del error cuadrático medio entre la imagen de entrada y la de salida.

Aunque los criterios de fidelidad objetiva ofrecen un mecanismo simple y conveniente, la mayoría de las imágenes descomprimidas acaban siendo vistas por seres humanos. En consecuencia, a menudo es más apropiado medir la calidad de la imagen mediante evaluaciones subjetivas de un observador humano. Esto se puede conseguir mostrando una imagen comprimida típica a un conjunto adecuado de observadores, promediando luego sus evaluaciones. Estas evaluaciones se pueden realizar empleando una escala absoluta de valores o bien por medio de comparaciones. A este tipo de evaluaciones se las conoce como *criterios de fidelidad subjetiva*.

## 5.4 MODELOS DE COMPRESIÓN DE IMÁGENES

Vamos a estudiar en este punto las características globales de un sistema de compresión de imágenes.

Este sistema consta de dos bloques estructurales distintos: un *codificador* y un *decodificador*. El codificador se alimenta con una imagen de entrada y crea un conjunto de símbolos a partir de los datos de entrada. Después de la transmisión a través de un *canal*, la representación codificada alimenta al decodificador, donde se

genera una imagen de salida reconstruida. Si el sistema está libre de errores la imagen reconstruida será una copia similar a la original, en cambio, si no lo es, decimos que el sistema presenta algún nivel de distorsión en la imagen reconstruida.

Tanto el codificador como el decodificador constan de dos funciones o sub-bloques relativamente independientes. El codificador está formado por un *codificador de fuente*, que elimina las redundancias (de codificación, entre píxeles y psicovisuales) de entrada y un *codificador de canal*, que aumenta la inmunidad al ruido de la salida del codificador de fuente. Dentro del codificador de fuente, las etapas que eliminan los distintos tipos de redundancias se denominan respectivamente *Codificador de Símbolos*, *Conversor* y *Cuantificador*. Como es de esperar, el decodificador incluye un *decodificador de canal* seguido de un *decodificador de fuente*. Si el canal entre el codificador y el decodificador está libre de ruido (sin error), se omiten el codificador y el decodificador de canal. Puesto que vamos a suponer nuestro medio de transmisión completamente libre de errores no vamos a incidir más en este caso.

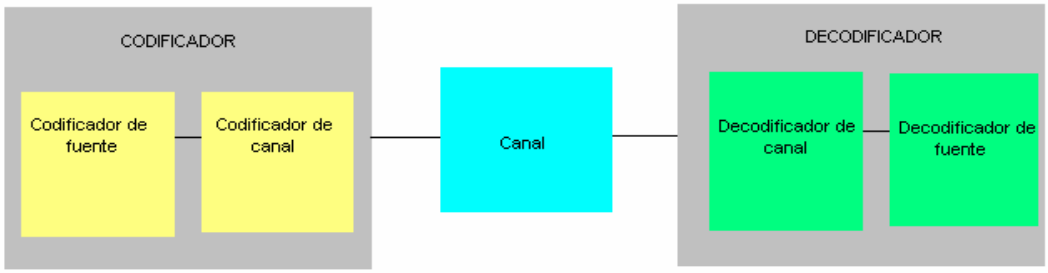



**Figura 5.2. Modelo de compresión de imágenes.**

Sin más preámbulo pasamos ya a describir las distintas técnicas de compresión que se han evaluado y estudiado para su posible implementación en este proyecto y

otras que aunque no se han tenido en cuenta sí que hemos considerado oportuna su inclusión dado su interés.

## 5.5 COMPRESIÓN SIN PÉRDIDAS

En numerosas aplicaciones, la compresión sin errores es la única forma aceptable de reducción de datos. Existen casos en que esta restricción está impuesta por ley, como es el caso de las imágenes médicas por ejemplo. En este proyecto fin de carrera, dada la naturaleza de la información transmitida y a la dificultad de la operación de detección de humo, que exige un procesamiento complejo de la imagen, se ha considerado que cualquier tipo de pérdida de información podría perjudicar de manera significativa al desarrollo de la mencionada actividad.

En la compresión sin pérdidas, cuando un conjunto de datos se comprime, se hace de forma que la imagen resultante de la descompresión produzca la imagen original exacta.

Este tipo de técnicas generalmente constan de dos operaciones relativamente independientes:

- Inventar una representación alternativa de la imagen en la que se reduzca la redundancia entre píxeles.
- Codificar la representación para eliminar la redundancia.

### 5.5.1 CODIFICACIÓN DE LONGITUD VARIABLE

El método más simple de compresión de imágenes sin errores consiste en reducir únicamente la redundancia de la codificación. Esta redundancia está normalmente presente en cualquier codificación binaria natural de los niveles de gris de una imagen. La forma de eliminar la información redundante es mediante la construcción de un código de longitud variable que asigne palabras de código más pequeñas a los niveles de gris más probables. Existen varias técnicas óptimas y casi óptimas de construcción de tal código y todas ellas se han evaluado como posibilidad a la hora de implantarlas en este proyecto. Es por ello por lo que a continuación procedemos a describirlas, detallando los pros y los contras de cada una y las razones que nos han llevado a rechazarlas o a aceptarlas.

### 5.5.2 CODIFICACIÓN DE HUFFMAN

Este algoritmo quizás sea el método de compresión más antiguo, aunque no por ello es el menos eficaz. Su inventor fue David Huffman, el cual lo publicó en 1951.

Cuando se codifican individualmente los símbolos de una fuente de información, la *Codificación de Huffman* consigue el número más pequeño posible de símbolos de código por símbolo de la fuente. El código resultante es óptimo para un valor fijo de n, con la restricción de que los símbolos se deben codificar uno a uno.

Este esquema de codificación comienza examinando el histograma de brillo de una imagen. Con este histograma la frecuencia de ocurrencia de cada nivel de brillo de la imagen está disponible. Una vez hecho esto, hemos de crear una serie de reducciones de la fuente ordenando las probabilidades de los símbolos considerados y combinando los símbolos de menor probabilidad en un único símbolo que los sustituye en la siguiente reducción de la fuente.

La segunda etapa del procedimiento de Huffman consiste en codificar cada fuente reducida, empezando por la fuente más pequeña, hasta llegar a la fuente original. Es decir, los códigos más cortos son asignados a los primeros (más frecuentes) valores m de la lista y eventualmente, los códigos más largos se asignan a los últimos (menos frecuentes) valores de la lista. Después, la imagen comprimida es creada simplemente sustituyendo los nuevos códigos de valores de brillo de longitud variable por los códigos de valores de brillo originales. Por supuesto, la lista de códigos Huffman que acopla los valores de brillo originales a sus nuevos códigos Huffman variables se debe añadir a la imagen para su uso en la operación de descompresión Huffman, como se muestra en la Figura 5.3.

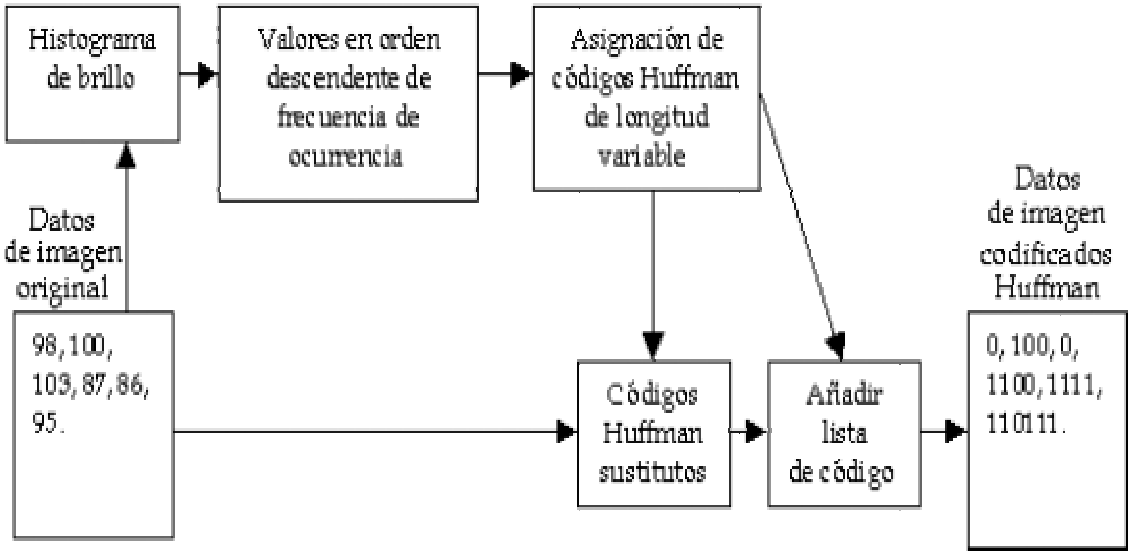



**Figura 5.3. Algoritmo de Huffman.**

Los códigos Huffman son asignados creando un árbol de Huffman que hace combinaciones con los valores de brillo basado en la suma de las frecuencias de ocurrencia. El árbol de Huffman asegura que los códigos más largos se asignen a los brillos menos frecuentes y los códigos más cortos se asignen a los brillos más frecuentes. Usando el brillo clasificado en orden de sus frecuencias de ocurrencia, los dos del final de la lista (menos frecuentes) se combinan y se etiquetan como 0 y 1. Los brillos combinados son representados por la suma de las frecuencias de ocurrencia. Entonces, se determinan y se combinan las próximas dos frecuencias de ocurrencia más bajas. De nuevo, el siguiente par se etiqueta 0 y 1 y es representado por la suma de las frecuencias de ocurrencia. Esto continúa hasta que todo el brillo se ha combinado. El resultado es un árbol que, cuando se sigue del final hasta el principio, indica el nuevo código Huffman binario para cada brillo en la imagen.

La Figura 5.4 muestra una imagen de 640 píxeles x 480 líneas, donde cada píxel es representado por simplicidad, por un valor de brillo de tres bits. El histograma de la imagen (Figura 5.5) muestra el número real de píxeles en la imagen con cada uno de los ocho valores de brillo. El brillo se ordena basado en sus frecuencias de ocurrencia y entonces se combina en un árbol de Huffman (Figura 5.6), como se describió anteriormente. Aunque todos los píxeles en la imagen original fueron codificados como valores de brillo de tres bits, los códigos Huffman son tan pequeños como un bit y pueden ser tan grandes como 7 bits. El código Huffman más

largo nunca puede ser mayor que el número de valores de brillo diferentes en la imagen (en este caso 8) menos 1. Aunque una imagen codificada con Huffman puede tener un poco de brillo con códigos muy largos, sus frecuencias de ocurrencia siempre son estadísticamente bajas.

**Figura 5.4. Imagen.**

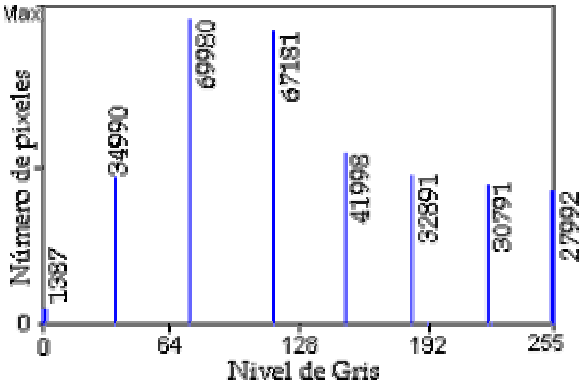



**Figura 5.5. Histograma de brillo de la imagen original.**

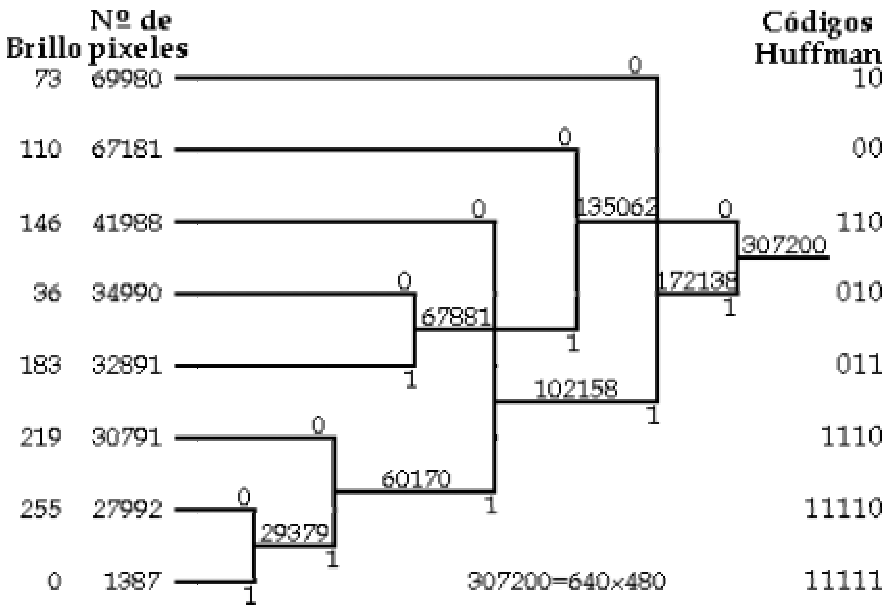



**Figura 5.6. Creación del árbol de Huffman.**

La cantidad de datos de la imagen original puede calcularse como 640x480x3 bits. La cantidad de datos de la imagen codificada por Huffman puede calcularse como la suma de las ocho frecuencias de ocurrencia multiplicadas por el respectivo número de bits en su código.

La descompresión de imágenes Huffman invierte el proceso de compresión sustituyendo los valores de brillo originales de longitud fija de un byte por valores codificados de longitud variable. La imagen original se reconstruye exactamente. El propio código es un código bloque de codificable instantáneamente de manera única. Se le conoce como código bloque porque cada símbolo de la fuente se corresponde con una secuencia fija de símbolos de código. Es instantáneo porque cada palabra de código de una cadena de símbolos de código se puede decodificar sin hacer referencia a los siguientes símbolos. Y también decimos que es decodificable de manera única, porque cualquier cadena de símbolos de código sólo se puede decodificar de manera única. La compresión de imágenes Huffman generalmente proporcionará razones de compresión de alrededor de 1.5:1 a 2:1.

Pero la codificación de Huffman pura presenta un alto grado de complejidad cuando se desean codificar un gran número de símbolos. Para el caso general en que tenemos J símbolos fuente, se deben realizar J – 2 reducciones de fuente así como J – 2 asignaciones de código. Por ello se proponen alternativas a esta forma de

codificar la información como por ejemplo la *Codificación de Huffman Truncada*, donde se codifican mediante Huffman sólo los K símbolos más probables de la fuente, siendo K un entero positivo que verifica que J > K. Para representar los restantes símbolos se utiliza un código prefijo seguido de un código de longitud fija apropiado. Evidentemente se trata de un algoritmo de codificación no óptimo.

### 5.5.3 CODIFICACIÓN ARITMÉTICA

Se usa en el estándar JBIG, y se basa en la existencia de infinitos números reales en el intervalo [0,1).

En la codificación aritmética no existe una correspondencia biunívoca entre los símbolos fuente y las palabras código. Es decir, este tipo de codificación no genera códigos de bloque. En cambio, se asigna una sola palabra código aritmética a una secuencia completa de símbolos fuente. La propia palabra código define un intervalo de números reales entre 0 y 1. Conforme aumenta el número de símbolos del mensaje, el intervalo utilizado para representarlo se va haciendo menor y se va incrementando el número de unidades de información necesarias para representar dicho intervalo. Cada símbolo del mensaje reduce el tamaño del intervalo según su probabilidad de aparición. Puesto que esta técnica no requiere, como sucedía con la técnica de Huffman, que cada símbolo de la fuente se traduzca en un número entero de símbolos del código (esto es, que los símbolos se codifiquen uno a uno), se alcanza (solo en teoría) el limite establecido por el teorema de codificación sin ruido.

La Figura 5.7 ilustra el proceso básico de la codificación aritmética. En este caso, se codifica una secuencia o mensaje de cinco símbolos, $a_1a_2a_3a_3a_4$, generados por una fuente de cuatro símbolos. Al principio del proceso de codificación, se supone que el mensaje ocupa todo el intervalo semiabierto [0,1). Como se muestra en la figura 5.8, este intervalo se subdivide inicialmente en cuatro regiones en función de las probabilidades de cada símbolo de la fuente. Por ejemplo, se asocia el subintervalo [0, 0.2) al símbolo $a_1$. Puesto que se trata del primer símbolo del mensaje a codificar, el intervalo del mensaje se reduce inicialmente a [0, 0.2). Así, en la Figura 5.7 el intervalo [0, 0.2) abarca toda la altura de la figura y se marcan los extremos con los valores del rango reducido. Posteriormente, se divide este rango reducido de acuerdo con las probabilidades de los símbolos de la fuente original, y

el proceso continúa con el símbolo del mensaje. De esta forma, el símbolo $a_2$ reduce el subintervalo a [0.04, 0.08), $a_3$ lo reduce aún más, dejándolo en [0.056, 0.072), y así sucesivamente. El último símbolo del mensaje, que se debe reservar como indicador especial de fin de mensaje, reduce el intervalo, que pasa a ser [0.06752, 0.0688). Por supuesto, se puede utilizar cualquier número que esté dentro del subintervalo, como por ejemplo el 0.068, para representar el mensaje.

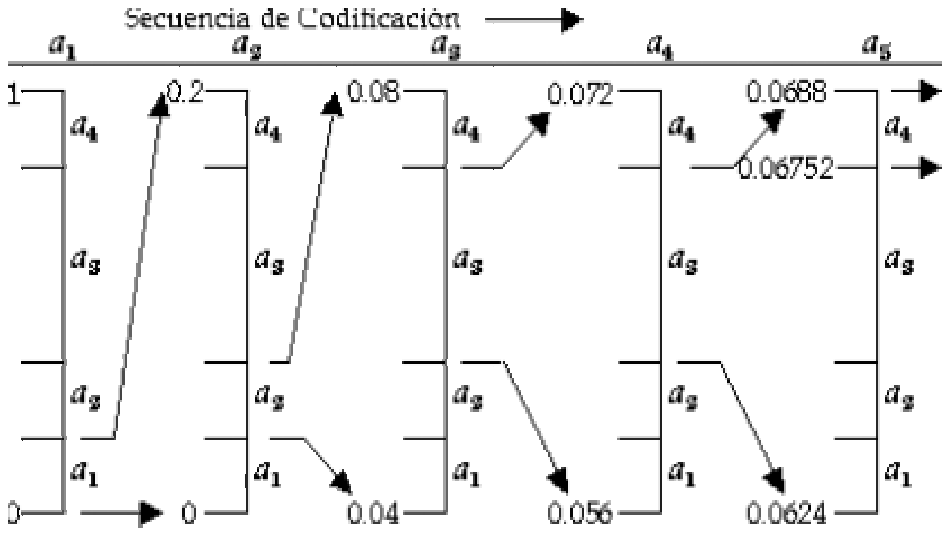



**Figura 5.7. Proceso de codificación aritmética.**

| Símbolo Fuente | Probabilidad | Subintervalo Inicial |
|---|---|---|
| $a_1$ | 0.2 | [0, 0.2) |
| $a_2$ | 0.2 | [0.2, 0.4) |
| $a_3$ | 0.4 | [0.4, 0.8) |
| $a_4$ | 0.2 | [0.8, 1) |

**Figura 5.8. Ejemplo de código de longitud variable.**

En el mensaje codificado aritméticamente de la Figura 5.7, se utilizan tres dígitos decimales para representar el mensaje de cinco símbolos. Esto se traduce en 3/5, ó 0.6 dígitos decimales por símbolo fuente, lo que se aproxima bastante a la entropía de la fuente, que resulta ser de 0.58 dígitos decimales por símbolo. Conforme aumenta la longitud de la secuencia a codificar, el código aritmético resultante se aproxima a límite establecido por el teorema de la codificación sin ruido. En la

práctica, existen dos factores que hacen que el rendimiento de la codificación se aleje de este límite:

- La inclusión del indicador de fin de mensaje, necesario para separar un mensaje de otro.
- La utilización de aritmética de precisión finita.

Los algoritmos implementados con este método de codificación han proporcionado tasas de compresión aceptables pero ninguno de ellos ha funcionado cerca de los tiempos que se le exigían a la compresión y a la descompresión.

### 5.5.4 CODIFICACIÓN POR ZONAS CONSTANTES

Un método sencillo pero efectivo para comprimir una imagen binaria o plano de bits consiste en utilizar palabras de código especiales que identifiquen grandes zonas de unos y ceros contiguos.

En este método la imagen se divide en bloques de m x n píxeles, que se clasifican como completamente blancos, completamente negros o de intensidad variable. A la categoría más probable o más frecuente se le asigna la palabra de código cero, de 1 bit. A las otras dos categorías se le asignan los códigos 10 y 11, de dos bits. Por supuesto, el código asignado a la categoría de intensidad variable se utiliza como prefijo, al que le siguen los m x n bits del bloque.

Este método propone múltiples variantes previendo las tendencias estructurales en la imagen a comprimir. Por ejemplo, si se va a codificar un documento de texto, sería más sencillo codificar las zonas blancas como cero y los restantes bloques (incluyendo los bloques negros) con un uno seguido de los bits que conforman cada bloque. De esta forma, como se esperan pocas zonas completamente negras, se agrupan junto con las regiones de intensidad variable, permitiendo el empleo de una palabra de código de un bit para los bloques blancos, altamente probables.

Otra variante efectiva de este método consiste en codificar las líneas totalmente blancas como ceros y las restantes líneas como unos seguidos de la secuencia normal de códigos WBS.

Pero el gran inconveniente que nos ha hecho declinar este método ha sido que tal y como hemos explicado, para que resulte efectivo es necesario tener una idea o unos conocimientos previos sobre la composición y contenido de la información a

comprimir. En este proyecto en cambio, no se dispone de demasiada información a priori sobre las imágenes que se van a procesar.

### 5.5.5 CODIFICACIÓN POR LONGITUD DE SERIES

En la compresión de imágenes sin pérdidas, hay una limitación intrínseca de cómo una imagen puede ser comprimida. La compresión más allá de este punto, eliminará alguna de la información necesaria para reproducir la imagen original, en su forma exacta.

La entropía de una imagen es una medida de este límite, es una medida de su contenido de información. Si la entropía es alta, la información de las imágenes tiende a ser altamente imprevisible. Mirándolo de otra manera, una imagen con información de alta entropía, contiene mucha aleatoriedad y baja redundancia. Si la entropía es baja, la información de las imágenes es más predecible, conteniendo una aleatoriedad pequeña y una redundancia alta.

Se puede calcular la entropía de una imagen como la probabilidad de su ocurrencia. Esto es, mostrándola como un número que representa el número de bits necesarios para representar esa probabilidad. Para cualquier imagen al azar, esto puede ser representado como:

*Entropía = Número de píxeles por fila x Número de líneas x Número de bits por píxel.*

Que, para 640 píxeles x 480 líneas x 8 bits por píxel, sería lo siguiente:

*Entropía = 640 x 480 x 8 = 2.457.600 bits.*

Esta es la entropía medida para cualquier imagen al azar de 640 x 480 x 8 bits. Para alguna de las $2^{2,457,600}$ posibles imágenes diferentes que pueden ser representadas por una imagen de estas dimensiones. A la inversa, hay una de $2^{2.457.600}$ posibilidades que una imagen en particular de estas dimensiones sea idéntica a otra.

Raramente se elaboran imágenes totalmente al azar con brillos que varíen constantemente, la entropía real de una imagen normal, generalmente será algo menor a la calculada anteriormente. Esto es debido a que la cantidad de datos de la imagen en bruto siempre será más alta que el promedio de la cantidad de datos de información. La entropía real de una imagen es la cantidad de información en

promedio de la imagen. En otras palabras, una imagen de 640 píxel x 480 línea x 8 bits puede ser comprimida desde una imagen inicial en bruto, requiriendo 2.457.600 bits, o un número más pequeño. La forma de compresión que efectúa esta codificación se llama codificación en entropía. La técnica de codificación en entropía reduce la redundancia de la imagen usando métodos de codificación de longitud variable. Esto se puede hacer a través de una codificación de números variables de píxeles con códigos de longitud fija, o codificando números fijos de píxeles con códigos de longitud variable.

La compresión de imágenes a través de la codificación por longitud de series se aprovecha del hecho de que varios píxeles cercanos en una imagen, estadísticamente tienden a tener el mismo valor de brillo. Esta forma de redundancia puede ser reducida por agrupación de píxeles de idéntico valor de brillo en un código simple.

El esquema en que trabaja la longitud de series es el siguiente. La imagen original es evaluada desde el primer píxel en la esquina superior izquierda, como se muestra en la Figura 5.9, mirando el primer píxel y sus vecinos siguientes a través de la línea, el esquema determina cuántos de los píxeles siguientes tienen el mismo valor de brillo. Si el próximo píxel o más, en la fila tiene el mismo valor de brillo que el primero, todos ellos son representados por un nuevo código. El nuevo código se compone de dos valores, un valor de brillo seguido por el número de píxeles (longitud de series) que tiene el mismo valor. El proceso se repite a los siguientes píxeles en la línea con un nuevo valor de brillo. Cuando se alcanza el extremo de la línea, el proceso empieza de nuevo al inicio de la próxima línea. El proceso continúa hasta que la imagen entera es codificada.

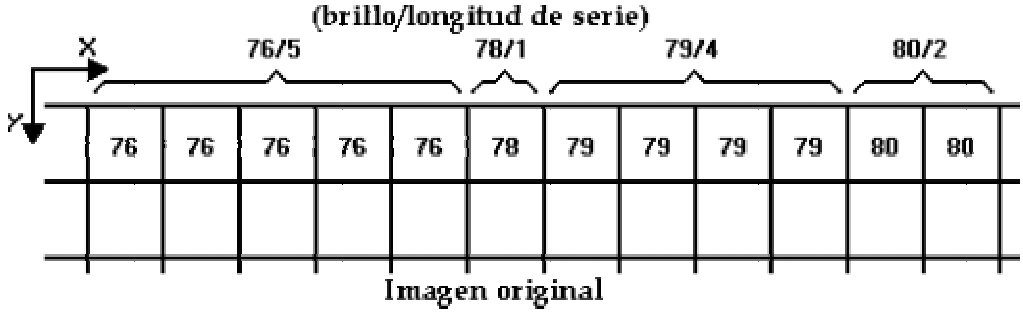



**Figura 5.9. Operación de una codificación por longitud de series.**

Si, por ejemplo, una secuencia de 15 píxeles tienen el mismo valor de brillo que es 234, ellos se codificarían en dos valores brillo / longitud de serie = 234 / 15. Los 15 bytes que se requieren para representar los 15 píxeles de brillo de la imagen original, están comprimidos en sólo 2 bytes. El resultado de comprimir la imagen por longitud de series es el de producir un código de 2 bytes, representando cada uno el valor de la serie de brillo y su longitud.

La compresión de imágenes a través de una longitud de series se descomprime expandiendo cada código de la imagen comprimida. El brillo que representa cada código se reproduce sobre la línea de la imagen descomprimida por el número de píxeles indicado en el valor de longitud asociado.

Esta técnica puede usarse en imágenes binarias con una mejora adicional. Debido a que los píxeles de brillo pueden ser sólo: 0 (negro) o 1 (blanco), entonces, no hay necesidad de codificar el valor de brillo por cada serie. En cambio, si se asume que la imagen empieza con un valor de brillo de 0, y simplemente se graba la longitud de serie del píxel hasta donde se encuentre un píxel blanco. Entonces, se graba la longitud hasta cuando un píxel negro sea encontrado, y así sucesivamente. Como el valor de brillo debe ser 0 o 1, no hay necesidad de presentar el valor de brillo explícitamente en la codificación por longitud de series de una imagen binaria. Esta modificación al funcionamiento de la longitud de series para las imágenes binarias proporciona factores de compresión muy altos.

En lugar de usar la técnica de longitud de series previamente discutida en imágenes con niveles de gris, se puede usar la técnica de longitud de series de la imagen binaria anterior en cada plano de bits de una imagen con niveles de gris. Esto es llamado codificación de planos de bits por longitud de series. De esta manera, se puede usar la eficacia de la técnica de longitud de series de una imagen binaria, mientras se trabaja con una imagen que contiene niveles de gris.

La compresión de imágenes utilizando longitud de series proporcionará factores de compresión de alrededor de 1.5:1 en imágenes que contienen niveles de gris. Factores de compresión de aproximadamente 4:1 o mayores de 10:1 pueden esperarse en las imágenes binarias, dependiendo de la imagen en particular.

Aplicando una codificación de planos de bits por longitud de series en imágenes que contienen niveles de gris generalmente producirán razones de compresión de 2:1.

### 5.5.6 CODIFICACIÓN Y TRAZADO DE CONTORNOS

La codificación por direcciones relativas es un método de representación de las transiciones de intensidad que conforman los contornos de una imagen binaria.

Otro método consiste en la representación de cada contorno mediante un conjunto de puntos del límite o mediante un solo punto del límite y un conjunto de vectores. Este último se conoce como *trazado directo de contornos*.

El método que se ha evaluado en este apartado hace uso de las características esenciales de los dos métodos anteriormente descritos y se llama *Cuantificación Diferencial Predictiva*. Este algoritmo es un procedimiento de trazado de contornos mediante exploración línea a línea.

En la cuantificación diferencial predictiva, se trazan simultáneamente los contornos frontal y trasero de cada objeto de una imagen para generar una secuencia de pares $(\Delta', \Delta'')$. El término $\Delta'$ es la diferencia de las coordenadas iniciales de los contornos frontales de líneas adyacentes y $\Delta''$ es la diferencia entre longitudes, medidas desde cada contorno frontal hasta su homólogo trasero. Estas diferencias junto con mensajes especiales que indican el comienzo de los nuevos contornos y el final de los viejos contornos, representan a cada objeto. Si se sustituye $\Delta''$ por la diferencia entre las coordenadas de los contornos traseros de líneas adyacentes, representada por $\Delta'''$, esta técnica se conoce como *Codificación de Doble Delta*.

Se ha descartado esta técnica dada la complejidad de proceso que representa cuando se trabaja con imágenes en color. Además este es un algoritmo poco usado y poco extendido en la industria y del que hemos encontrado poca documentación. Esto nos lleva a dudar de su efectividad.

### 5.5.7 CODIFICACIÓN PREDICITIVA SIN PÉRDIDAS

Este método se basa en la eliminación de las redundancias entre píxeles muy próximos, extrayendo y codificando únicamente la nueva información que aporta cada píxel. Se define la nueva información de un píxel como la diferencia entre el valor real y el valor estimado de ese píxel.

En las Figuras 5.10 y 5.11 se muestran los componentes de un sistema de codificación predictiva sin pérdidas. Podemos apreciar la presencia de un codificador y de un decodificador y ambos contienen un predictor idéntico. A medida que se va introduciendo sucesivamente cada píxel de la imagen de entrada, representado por $f_n$, en el codificador (Figura 5.10), el predictor genera el valor anticipado de dicho píxel en función de algún número de entradas anteriores. La salida del predictor se redondea después al entero más cercano, representado por $f'_n$, y se utiliza para construir la diferencia, o error de predicción:

$$e_n = f_n - f'_n$$

Que se codifica utilizando un código de longitud variable (por medio de un codificador de símbolos) para generar el siguiente elemento del flujo de datos comprimidos. El decodificador de la Figura 5.11 reconstruye $e_n$ a partir de las palabras código de longitud variable y realiza la operación inversa:

$$f_n = e_n + f'_n$$

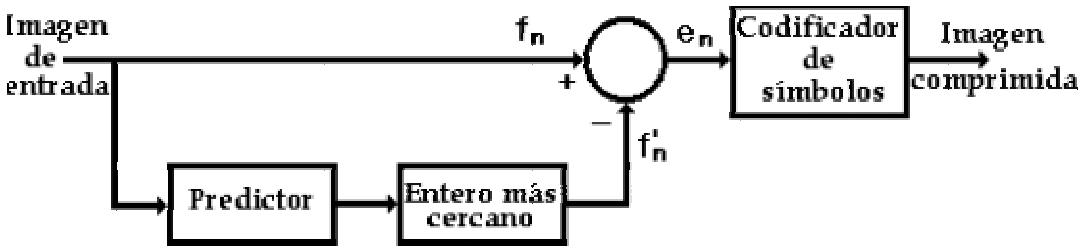



**Figura 5.10. Codificador.**

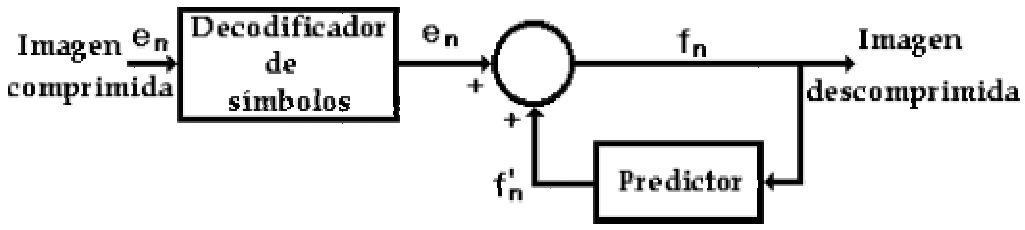



**Figura 5.11. Decodificador.**

En la codificación predictiva de dos dimensiones, la predicción es una función de los píxeles anteriores de una exploración de izquierda a derecha y de arriba abajo de una imagen.

La estructura fundamental para la codificación predictiva de una imagen, es la modulación de pulsos codificados diferenciales (DPCM). Esto implica, que la cantidad que se codifica es la diferencia entre píxeles de brillo.

El esquema de compresión DPCM opera en la imagen completa, píxel por píxel. El primer píxel, en la esquina superior izquierda de la imagen, permanece inalterado; este es exactamente codificado con su brillo original. El proceso se mueve ahora al segundo píxel en la línea, donde el siguiente valor de brillo del píxel se sustrae de los actuales píxeles de brillo. El resultado de la sustracción es el nuevo valor codificado para el segundo píxel en la imagen. Este proceso se repite por toda la línea. Al inicio de la próxima línea, el proceso comienza de nuevo, y este continúa hasta que la imagen entera es codificada. Las operaciones de compresión y descompresión de la codificación predictiva sin perdidas se muestran en la Figura 5.12.

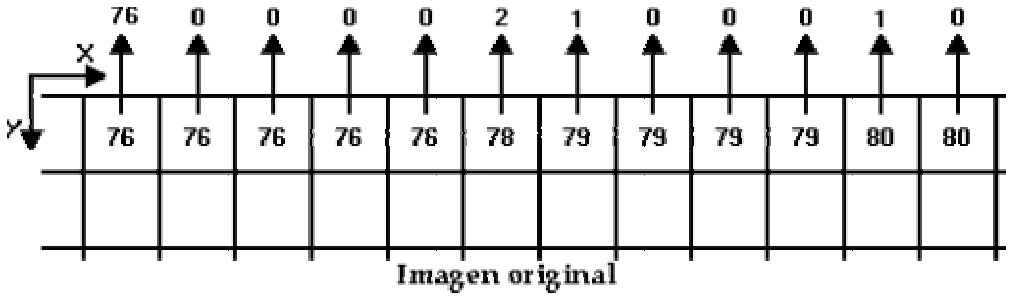



**Figura 5.12. Operación de una codificación predictiva sin pérdidas.**

Como ejemplo, se toman los cinco primeros píxeles de la línea de una imagen que contiene los siguientes valores de brillo: 23, 48, 76, 56, y 83. Se asumirá que la imagen fue originada con valores de brillo de 8 bits. Los valores DPCM codificados son mostrados en la figura 5.13.

| Imagen original | Valores de 8 bits | Código DPCM de 6 bits |
|---|---|---|
| Píxel # 1 | 23 | 23 |
| Píxel # 2 | 48 | 48-23=25 |
| Píxel # 3 | 76 | 76-48=28 |
| Píxel # 4 | 56 | 56-76=-20 |
| Píxel # 5 | 83 | 83-56=27 |
| Total de bits | 8x5=40 bits | 6x5=30 bits |

**Figura 5.13. Codificación DPCM sin pérdidas.**

Los cinco primeros píxeles de brillo son comprimidos desde valores de 5 x 8 bits = 40 bits a valores de la diferencia de 5 x 6 bits = 30 bits.

El esquema de compresión DPCM trabaja con la suposición de que los píxeles vecinos serán similares o altamente correlacionados. Como resultado, sus diferencias normalmente serán valores muy pequeños. Mirando los valores en el ejemplo anterior, ninguno es mayor de 31 o menor de -32, éstas diferencias de valor se pueden codificar usando números de 6 bits en lugar de números de 8 bits, permitiendo un factor de compresión de 8/6=1.333:1. Si todos los valores de las diferencias estuvieran debajo de 16, sólo serían necesarios números de 4 bits, permitiendo un factor de compresión de 2:1.

La operación de compresión en DPCM trabaja mejor en imágenes que no tienen un número desmesuradamente grande de brillo que oscila entre píxeles adyacentes. Cuando se aplica a imágenes normales, la codificación DPCM pueden proporcionar factores de compresión alrededor de 2:1. Para las imágenes con series largas de valores de píxeles constantes, los factores de compresión se pueden incrementar significativamente.

El motivo que nos ha llevado a no implantar este método es que supone a priori demasiadas características de la información a transmitir. Por ejemplo, tal y como se ha explicado anteriormente, el esquema de compresión DPCM trabaja con la suposición de que los píxeles vecinos serán similares o altamente correlacionados y además trabaja mejor en imágenes que no tienen un número desmesuradamente grande de brillo que oscila entre píxeles adyacentes. En cambio, como venimos desarrollando a lo largo de este documento, nuestro proyecto fin de carrera no

presupone ningún tipo de características en la información a transmitir. Es por ello por lo que al transmitir todo tipo de imágenes este método resulta ineficiente.

### 5.5.8 ALGORITMO LEMPEL-ZIV

Tuvo sus orígenes en 1977, cuando Abraham Lempel y Jacob Ziv (LZ) publicaron un artículo describiendo una técnica nueva: *uso de un diccionario para comprimir*. Estos mismo autores publicaron otro artículo en 1978, describiendo un perfeccionamiento del algoritmo LZ77, llamado en este caso LZ78, y que ha dado lugar a un algoritmo muy popular: LZW, modificación del LZ78 propuesta por Terry Welch. El conjunto de estos algoritmos y sus modificaciones constituyen las técnicas de diccionario.

El algoritmo LZ77 consiste en usar una *ventana* dividida en dos mitades, que no tienen porqué ser del mismo tamaño. Típicamente, ambas son de tamaño potencia de 2, siendo la primera mucho mayor que la segunda (por ejemplo, 8192 frente a 32). El algoritmo para comprimir busca ocurrencias de la segunda mitad de la ventana en la primera, y en vez de codificar la ocurrencia entera, almacena solamente el desplazamiento dentro de la primera mitad y la longitud de esa ocurrencia. En caso de que la segunda mitad no se encuentre dentro de la primera, se almacena una secuencia de símbolos sin comprimir, se desplaza la ventana, y se sigue el proceso.

Para que el descompresor pueda distinguir cuándo está leyendo una secuencia de símbolos sin comprimir ó un desplazamiento más una longitud, se les antepone un bit que hace las veces de separador:

- Para codificar un símbolo de la segunda mitad que no aparece en la primera, se le antepone un bit a 0 antes del símbolo. Por lo tanto, el número de bits que se gasta en este proceso es fijo y conocido: 1+8 = 9, puesto que los símbolos ASCII usan 8 bits, esto es, 1 byte.

- Para codificar una ocurrencia de la segunda mitad en la primera se antepone un bit a 1, con lo que almacenamos 1 + desplazamiento + longitud. Los bits que se usan para codificar el desplazamiento dependen del tamaño de la primera mitad y los de la longitud del de la segunda mitad. En el ejemplo anterior los tamaños eran 8192 y 32, entonces, se usarían 13 bits para el desplazamiento y 5 para la longitud, puesto que 2^13=8192 y 2^5=32.

Existen muchas modificaciones y mejoras que se han aplicado a este método entre las que destacamos las siguientes:

- *LZ77 optimizando el campo longitud de coincidencia*: Se basa en el algoritmo anteriormente expuesto, y contempla una optimización del campo longitud de coincidencia. En el ejemplo del método LZ77 puro, se obtenía que se debían usar 5 bits para codificar la longitud de la coincidencia de la segunda mitad de la ventana en la primera. Esto puede llegar a ser altamente ineficiente si las longitudes que aparecen en la información a comprimir son bastante más bajas que el tamaño de la segunda mitad de la ventana. En el ejemplo, usaríamos 5 bits para almacenar longitudes del orden de 2 bits por ejemplo. En cambio si insertamos un bit antes de almacenar el campo longitud que indicará si se almacena con los bits correspondientes al tamaño de la segunda mitad (5 en el ejemplo) ó con un número de bits menor que puede fijar el usuario y que puede ser modificado en tiempo de ejecución. Es decir, en vez de codificar como 1 + desplazamiento + longitud, codificaríamos como: *1 + desplazamiento + 0 + longitud*, si longitud es mayor de un umbral y *1 + desplazamiento + 1 + longitud*, si longitud es menor de ese umbral. El umbral citado antes no es más que $2^{número}$ de bits de optimización. Si usásemos por ejemplo 2 bits de optimización, para una coincidencia de longitud 3, se gastarían 1+13+1+2=17 bits, mientras que con el método LZ77 puro gastaríamos 1+13+5=19 bits. Puede parecer poco el ahorro, pero el secreto de esta optimización reside en la posibilidad de adaptar el número de bits de optimización al número que minimice la función de gasto. En un fichero de texto, por ejemplo, el ratio de compresión típicamente suele estar en torno al 60% - 70%, usando el método LZ77 puro. Si se usa esta optimización, éste se rebaja en torno al 35% - 45%, compresión muy cercana al 25% del algoritmo ZIP, descrito más adelante, y teniendo en cuenta que los ficheros son de texto.

- *Algoritmo ZIP*: Es la modificación del método LZ77 que más se usa en estos momentos. Usa como base el método LZ77 puro, pero codifica una coincidencia usando el método Huffman puro. Con ello consigue una mayor potencia de compresión, que en muchos casos supera levemente a la del conocido algoritmo LZW. La implementación ZIP propuesta por el GNU fija el tamaño de la primera mitad de la ventana a 32768( 2^15) y el tamaño de la segunda a 258, usando

algunas longitudes como códigos de escape para notificar al descompresor. Usa un árbol de Huffman para codificar el campo desplazamiento y otro diferente para el campo longitud. La compresión se lleva a cabo por bloques de 65536 bytes, con lo que al final de cada bloque se conocen las frecuencias de aparición de los símbolos a codificar, pudiéndose usar el método Huffman puro y no ser necesario usar el Adaptativo que se describe a continuación.

Las pruebas realizadas con este método de compresión y sus distintas modificaciones, han proporcionado buenos resultados en cuanto a la tasa de compresión. En cambio todos los códigos implementados han sido excesivamente lentos dada la complejidad de proceso que este algoritmo presenta. Ello ha supuesto su descarte puesto que la transmisión de imágenes sin ningún tipo de compresión se realizaba de manera mucho más rápida que con cualquiera de los algoritmos que implementaban este método.

### 5.5.9 MÉTODO DE HUFFMAN ADAPTATIVO

Es una modificación del algoritmo de Huffman puro descrito en el apartado 5.5.2, propuesta por Faller y Gallager y es similar en cuanto a la asignación de palabras código de longitud dependiente de la frecuencia de aparición. Puesto que ésta no se conoce de antemano, se usa un algoritmo que usa la frecuencia conocida hasta ese momento.

Se han introducido conocidas mejoras a este método de Huffman adaptativo, como son la propuesta por Knuth en 1985 (cuyo resultado se denomina *Algoritmo FGK*) y la propuesta por Vitter en 1987. Estas se estudian más adelante en este apartado.

La extensión del algoritmo de Huffman puro se hace necesaria por las siguientes razones:

- Eran necesarios conocimientos de los estadísticos de la información a transmitir y no en todos los casos esta está disponible.
- Aún cuando los estadísticos de la información estuvieran disponibles, puede ser complicado cuando hay muchas tablas que enviar si estamos utilizando un modelo de orden distinto de cero, el llevar la cuenta de las incidencias de cada

símbolo y la probabilidad de ocurrencia de los símbolos que se esperan a continuación.

Este algoritmo realiza un mapeo de los mensajes de la fuente de información para realizar la codificación basándose en una estimación dinámica de las probabilidades de los símbolos de la fuente. La codificación se realiza de forma adaptativa, cambiando de manera que permanezca óptima para la siguiente estimación. En resumidas cuentas podemos decir que el codificador aprende las características de la fuente de información. Por su parte, el decodificador debe aprender a la vez que el codificador para actualizar continuamente su árbol de Huffman de forma que tanto codificador como decodificador permanezcan en perfecta sincronía.

Una de las principales ventajas de este método consiste en que solamente requiere una pasada por los datos que se pretenden comprimir. Ello agiliza de manera importante el proceso de compresión proporcionando tiempos de funcionamiento apropiados para nuestro trabajo.

Este algoritmo puede trabajar a un nivel global de la información pero es mucho más efectivo cuando trabaja a un nivel local, ya que el árbol está constantemente evolucionando. Por ejemplo, si un archivo comienza con una secuencia de un mismo carácter que no se repite de nuevo a lo largo de dicho archivo, en el algoritmo de codificación estática este carácter estará en la parte baja del árbol porque aparece pocas veces en relación al archivo completo, de esta forma se codificará con un número grande de bits. En cambio en el algoritmo adaptativo, este carácter se colocará en la hoja más alta del árbol para su decodificación, porque ha sido empujado durante el proceso de decodificación por caracteres más frecuentes.

Pero el fundamento del método de Huffman Adaptativo radica en la propiedad de *hermandad* que establece que cada nodo excepto la raíz tiene un hermano y estos nodos pueden ser listados en orden inversamente proporcional al incremento del peso con cada nodo adyacente a sus hermanos. Para mantener esta propiedad, realizamos un seguimiento del orden de cada nodo y sus pesos correspondientes donde el orden se usa simplemente como un sistema de numeración simple para los pesos que se incrementa de izquierda a derecha, de arriba abajo y donde el peso significa el número de veces que el valor almacenado en ese nodo ha aparecido en ese archivo. Los nodos que no contienen ningún valor (como la raíz y otros nodos

internos) tienen un peso igual a la suma de los valores de los pesos de sus dos nodos hijos. Nodos con un orden mayor tendrán también pesos mayores.

Incluimos en la figura 5.14 la construcción de un árbol como ejemplo:

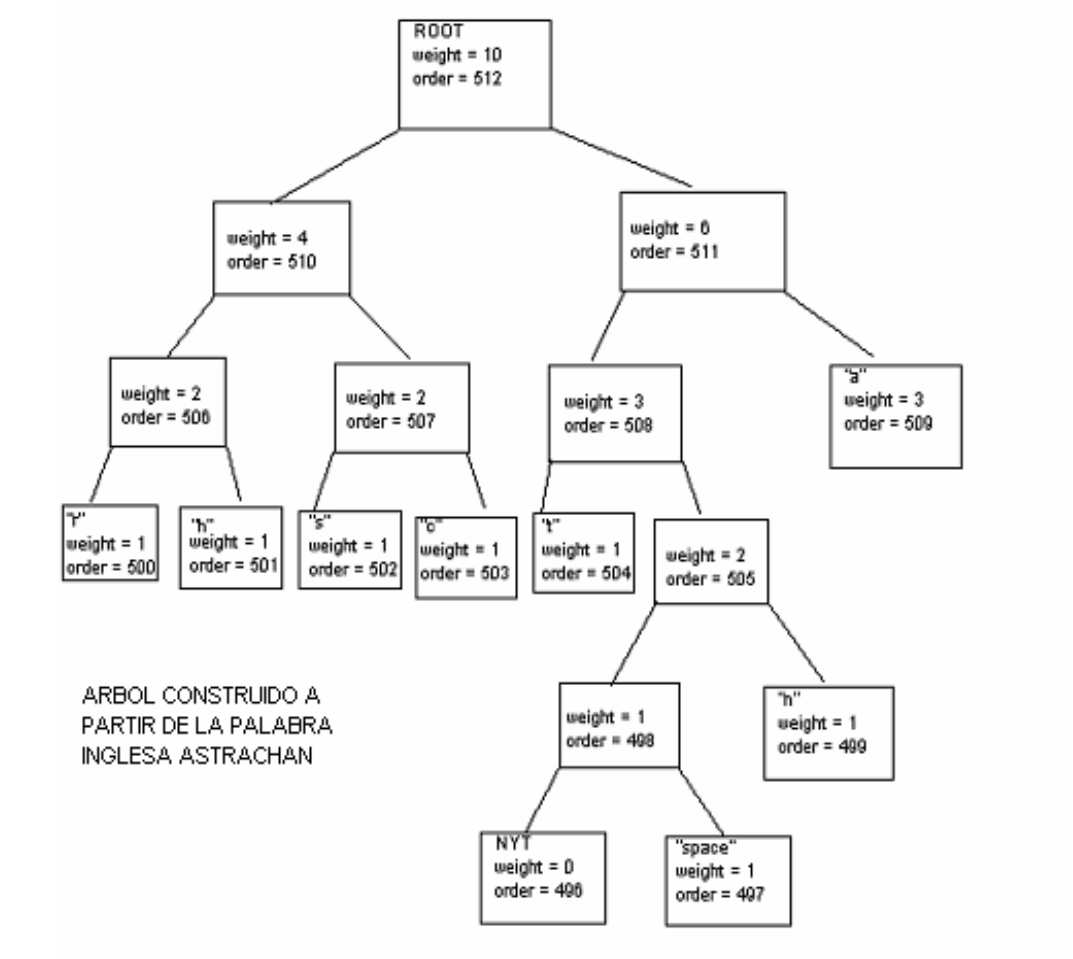



**Figura 5.14. Ejemplo de construcción del árbol de Huffman.**

La ordenación comienza con el valor 512 porque hay 256 caracteres que se pueden representar con 8 bits y un número máximo de nodos internos igual a 255.

A la hora de realizar la manipulación de estos árboles se ha de tener en cuenta que el contenido del árbol es modificado a la vez que se lee el fichero. Se deben mantener las siguientes propiedades:

- Cada nodo tiene un hermano.
- Nodos con pesos más altos también tienen órdenes más altos.

- En el mismo nivel del árbol, el nodo situado más a la derecha será el que tenga el valor de peso más alto, aunque pudiera haber otros nodos con el mismo valor.
- Las hojas de cada nodo contienen el valor de un símbolo que proviene de la fuente, a excepción del nodo NYT (Not Yet Transmitted), que será el nodo al se añadirán todos los nuevos caracteres.
- Los nodos internos contienen valores de pesos iguales a la suma de los pesos de todos sus nodos hijos.
- Todos los nodos que tengan el mismo peso estarán colocados en orden consecutivo.

Cada árbol contiene un nodo raíz y un nodo NYT, donde el nodo NYT posee el orden más bajo del árbol. Cuando leemos un símbolo del archivo fuente, se comprueba en el árbol si ya existe este carácter. Si no existe, del nodo NYT brotan dos nuevos nodos. El nodo situado a la derecha contiene el carácter leído de la fuente y el de la izquierda es el nuevo nodo NYT. En cambio, si el carácter obtenido ya se encontraba en el árbol, simplemente se actualiza el peso del nodo que contenía el carácter en cuestión. En algunos casos, cuando el nodo no es el nodo de mayor orden para su peso, necesitaremos intercambiar ese nodo para poder cumplir la propiedad de que mayores pesos corresponden a nodos con mayor orden. Para realizar este intercambio tendremos antes de actualizar el peso del nodo en cuestión, buscar en el árbol los nodos que presenten igual peso y permutarlo con el nodo de mayor orden e igual peso. Finalmente se realiza la actualización de dicho peso.

Hemos de tener presente que al modificar los valores de los pesos de un determinado nodo, el cambio afectará a todos los nodos situados por encima de éste. Por consiguiente, después de insertar un nodo debemos comprobar el valor de los nodos padres siguiendo el mismo procedimiento que seguimos cuando procesamos un símbolo que ya ha aparecido anteriormente.

Antes de realizar un intercambio de nodos hemos de hacer las siguientes comprobaciones:

- El nodo raíz nunca debe intercambiarse.

- Como estamos procesando el árbol hacia arriba, los nodos por encima del nodo que estamos procesando no se actualizan. Por consiguiente debemos asegurarnos que nunca intercambiamos un nodo con sus padres.

- Aunque los punteros deben intercambiarse en el árbol, hemos de asegurarnos que se inicializa el orden para encajarlo en la nueva configuración. El orden no es una medida relacionada con el valor de un nodo, sino con su posición dentro del árbol.

Presentamos a continuación un diagrama de bloques que representa esquemáticamente el proceso que se realiza sobre los nodos del árbol:

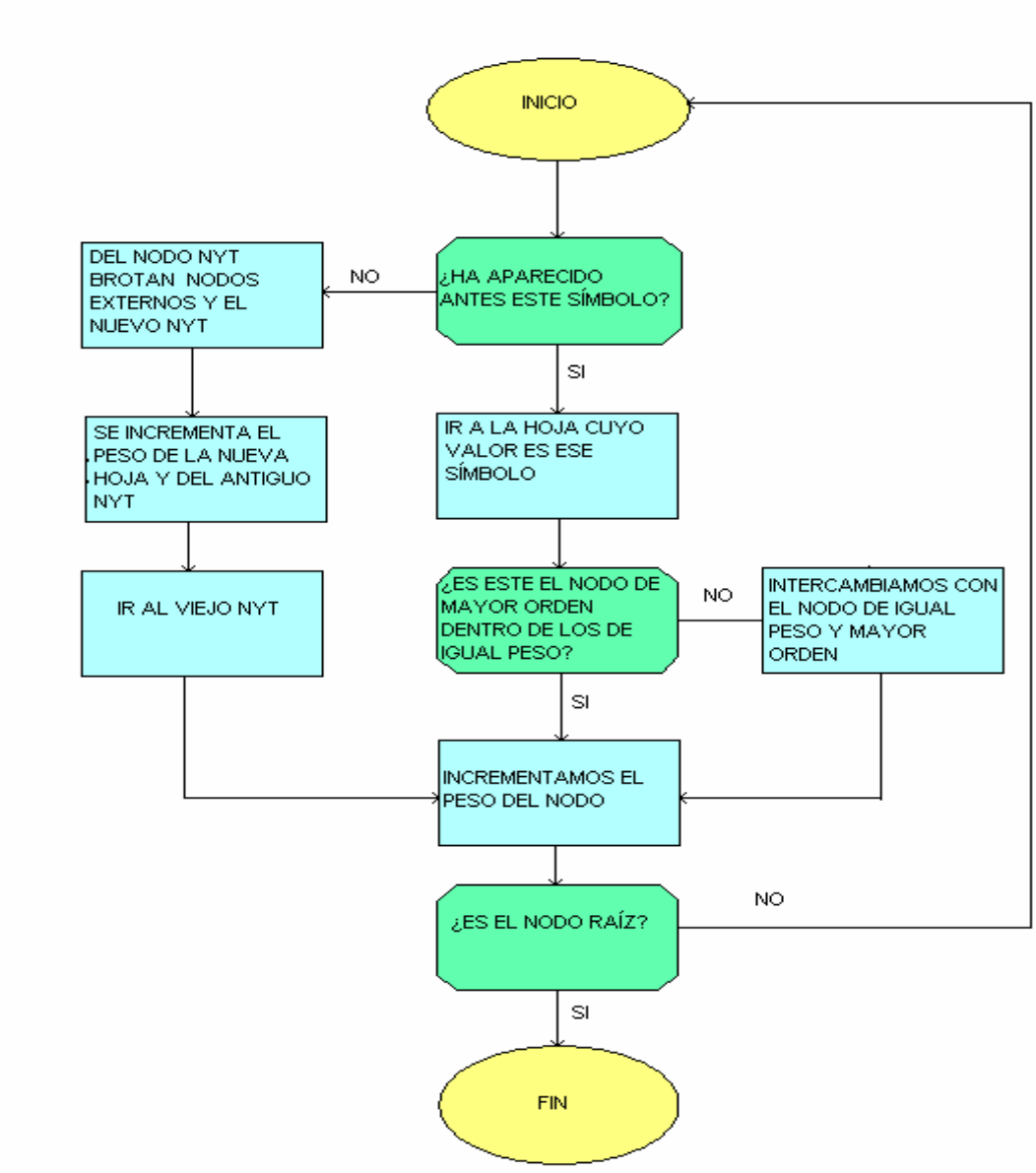



**Figura 5.15. Proceso de creación del árbol de Huffman Adaptativo.**

El proceso de codificación una vez construido correctamente el árbol es bastante sencillo. Simplemente hemos de leer carácter a carácter los símbolos del archivo que se pretende comprimir. Si el símbolo se ha leído antes en otra posición dentro de ese mismo archivo lo escribimos en el archivo de salida (archivo comprimido), siguiendo en camino desde la raíz hasta la posición de la hoja donde se encuentre este valor, escribiendo un 1 por cada movimiento a la derecha y un 0 por cada movimiento a la izquierda. Si se trata de un carácter nuevo, escribimos en el archivo de salida el camino desde la raíz hasta la hoja que contiene el NYT para indicarle al decodificador que a continuación viene un nuevo carácter. De esta forma se escribe en la salida el nuevo carácter en sí mismo en el archivo decodificado. Finalmente se actualiza el árbol con los nuevos valores del carácter entrante.

En la figura 5.16 se muestra el diagrama de bloques del procedimiento:

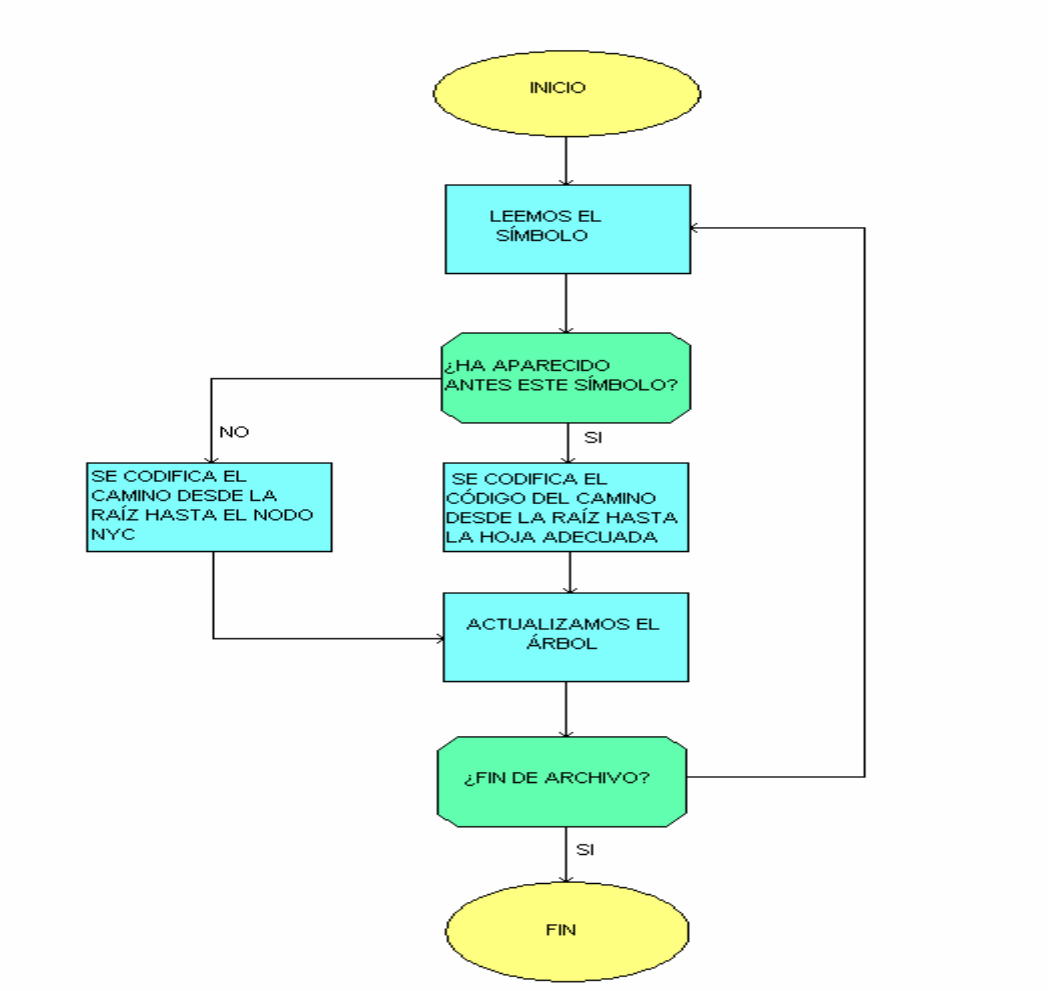



**Figura 5.16. Codificación de Huffman Adaptativo.**

Con respecto al proceso de decodificación, podemos decir que es muy similar a la codificación. Hemos de leer un bit cada vez, recorriendo el árbol hasta que lleguemos, por ejemplo a una hoja, en cuyo caso escribiremos en el archivo de salida los 8 bits que corresponden al carácter que se encuentra representado en esa hoja. Una vez hecho esto se actualiza la cuenta de este carácter en el árbol, asegurándonos que se realizan los cambios pertinentes en el árbol completo. Si la hoja a la que llegamos recorriendo el grafo es el nodo NYT, leemos los siguientes nueve bits que vienen a continuación y se escriben los 8 bits correspondientes al carácter que corresponde. Tras esto se inserta el nuevo carácter en el árbol.

Es muy importante entender que el compresor y el descompresor, aunque leen los caracteres de distinta manera, deben construir árboles exactamente iguales. En un momento dado del proceso en ambas operaciones, los árboles serían exactamente iguales si se compararan. Pero no nos confundamos y pensemos que codificador y decodificador trabajan simultáneamente. No trabajan a simultáneamente pero construyen los mismos árboles tras haber leído la misma información.

El diagrama de bloques del proceso de decodificación se muestra en la figura 5.17:

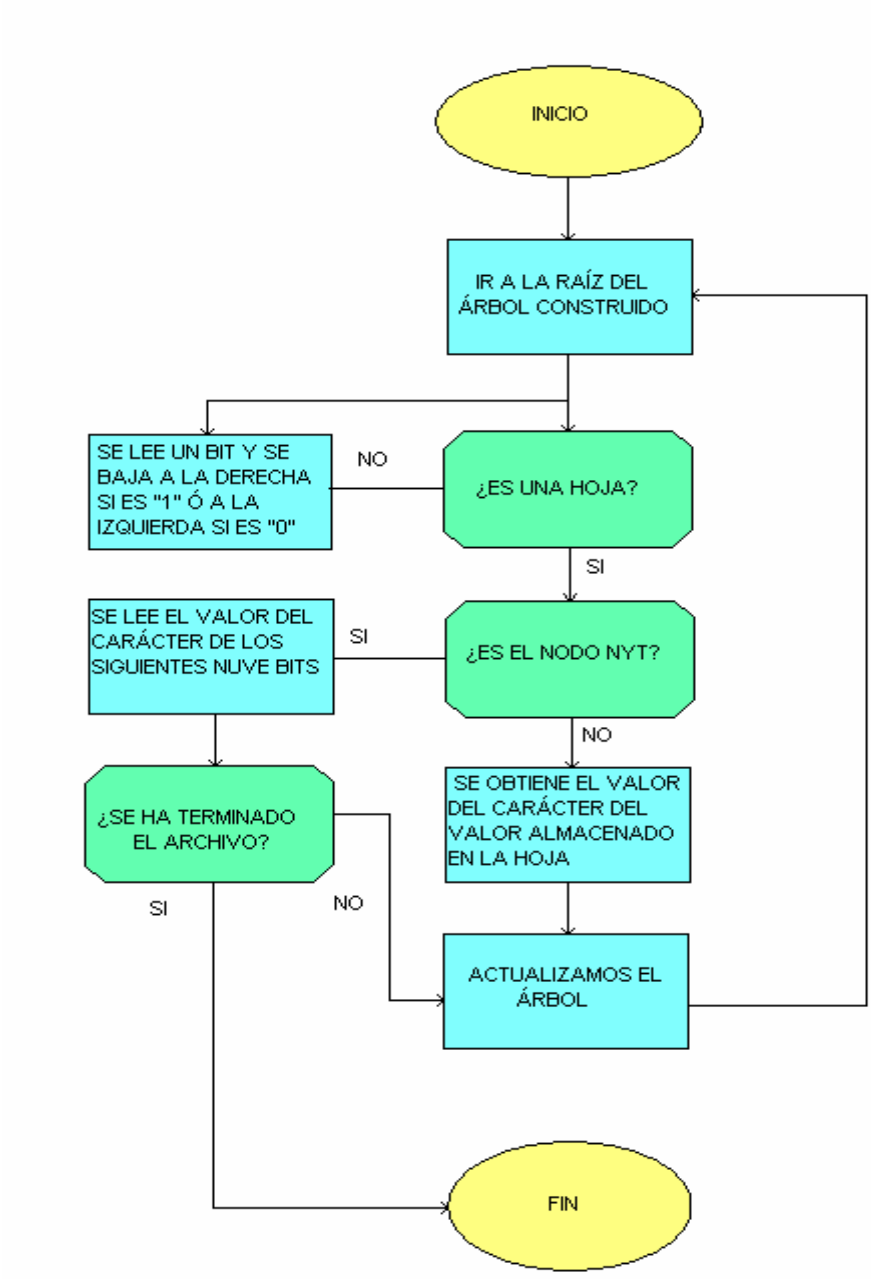



**Figura 5.17. Decodificación de Huffman Adaptativo.**

En cuanto a la implementación del algoritmo diremos que existen varios métodos para hacerlo. Entre todos los posibles destacan el FGK y el método Vitter. El algoritmo FGK funciona razonablemente bien y es muy sencillo de implementar, pero se encuentra muy por debajo del algoritmo de Vitter en lo referente a la minimización del tamaño del árbol y a la rapidez para encontrar el camino raíz-hoja. Este algoritmo de Vitter mantiene el tamaño del árbol en un mínimo absoluto pero es difícil de implementar para que el programa funcione a una velocidad razonable.

En líneas generales podemos describir los métodos de la siguiente manera:

- *FGK*: Este algoritmo, al contrario de lo que ocurre con el algoritmo de Vitter, no comprueba el árbol entero cuando se actualiza la clase de peso ni utiliza un esquema de orden numerado. Solamente compara el nodo con su hermano de la derecha o el hermano de la derecha de sus padres. Esto provoca que el árbol no sea lo más pequeño posible pero en cambio acelera de manera considerable el proceso de actualización hoja-raíz. La forma de proceder en este caso es:
  - Insertamos el elemento en el árbol.
  - Incrementamos el peso del nodo, siguiendo la numeración y la estructura marcada por el camino y el peso.
  - Comparamos el peso con su hermano derecho y se permuta si procede.
  - Nos movemos al padre.
  - Comparamos los pesos con los hijos de ese padre y realizamos la permuta si procede.
  - Seguimos el camino de la hoja a la raíz, teniendo en cuenta el peso del nuevo nodo.
  - Continuamos el seguimiento del camino, permutando pesos con los hermanos de la derecha cuando proceda.
- *Vitter:* Este algoritmo es fruto de las investigaciones del científico informático Jeffrey Vitter. Este método es el que se ha venido describiendo a lo largo de este apartado. La principal dificultad que presenta es que es difícil hacerlo trabajar rápido ya que para actualizar el árbol se examina cada uno de los nodos que componen el árbol con el fin de encontrar aquellos que presentan similar peso. Podemos citar dos formas de realizar estas tareas. La primera y más simple consiste en trabajar con un vector donde almacenar los elementos y poder acceder a ellos de forma indexada. Operando de esta forma todos los nodos con el mismo peso deberán estar en posiciones consecutivas en este vector. Podemos tener también otro vector en el que guardemos los valores de los caracteres presentes en las hojas indexado por el valor del carácter presente en la hoja. Este resultará especialmente útil a la hora de comprobar si un valor ha sido ya o no

insertado en el árbol. Pero la forma de agilizar de manera considerable estos procesos consiste en utilizar una estructura de datos de *árbol flotante*. Esta estructura de árbol flotante consta únicamente de vectores y mapas para implementar el algoritmo y dicho árbol mantiene punteros al hijo derecho y a los padres solamente.

Como conclusión a la descripción del método de Huffman Adaptativo diremos que ha sido el elegido para su implementación en este proyecto fin de carrera por los siguientes motivos:

- Es un método simple y del que se dispone suficiente documentación.
- Es un método de compresión sin pérdidas.
- Realiza la compresión y la descompresión en unos tiempos bajos. Es decir, funciona con la rapidez adecuada.
- Alcanza tasas de compresión aceptables, que a la larga conseguirán una razonable descarga de la red.

## 5.6 COMPRESIÓN CON PÉRDIDAS

Tal y como se explicó al principio del capítulo, la compresión con pérdidas nunca ha sido una posibilidad de diseño en este proyecto fin de carrera, ya que la ausencia de pérdidas era una clara especificación de este trabajo. Aún así, dada la relevancia de algunos de los métodos que hemos encontrado durante el proceso de investigación, consideramos oportuno incluir una breve introducción a estos métodos más importantes, con el fin de que el lector tenga una idea de la forma en la que operan este tipo de técnicas.

### 5.6.1 INTRODUCCIÓN

La compresión con pérdidas se basa en la idea de comprometer la precisión de la imagen reconstruida con el fin de lograr una mayor compresión. Si se puede tolerar la distorsión resultante (que puede ser o no visualmente aparente), el incremento del nivel de compresión puede ser significativo. De hecho, muchas técnicas de codificación con pérdidas son capaces de reproducir imágenes monocromas reconocibles a partir de datos que se han comprimido con un factor de 30:1 y las imágenes son virtualmente indistinguibles de las originales con factores de 10:1 y

20:1. Sin embargo en la codificación sin errores de una imagen monocroma, rara vez se consigue una reducción de los datos superior a 3:1.

Describimos a continuación los métodos más importantes y más utilizados a la hora de realizar compresión de imágenes con pérdidas.

### 5.6.2 EL ESTÁNDAR JPEG

A fin de proporcionar un estándar universal para la compresión mínima, el Grupo de Expertos Fotográficos Asociados o Joint Photographic Experts Group (JPEG) desarrolló un formato de almacenamiento de la imagen digital basado en estudios de la percepción visual humana. El estándar JPEG describe una familia de técnicas de compresión de imágenes fijas de tonalidad continua en escala de grises o color (24 bits). Sin embargo, numerosas aplicaciones han usado la técnica también para compresión de video, porque proporciona descompresión de imagen de calidad bastante alta a una razón de compresión muy buena, y requiere menos poder de cálculo que la compresión MPEG (Motion Pictures Experts Group).

Debido a la cantidad de datos involucrada y la redundancia psicovisual en las imágenes, JPEG emplea un esquema de compresión con pérdidas basado en la codificación por transformación. El estándar resultante tiene tantas alternativas como sean necesarias para servir a una amplia variedad de propósitos y hoy día es reconocido por la Organización Internacional de Estándares con el nombre de ISO 10918.

El estándar JPEG define tres sistemas diferentes de codificación:

- Un sistema de codificación básico, con pérdidas, que se basa en la Transformada Discreta del Coseno y es apropiado para la mayoría de las aplicaciones de compresión.
- Un sistema de codificación extendida, para aplicaciones de mayor compresión, mayor precisión, o de reconstrucción progresiva.
- Un sistema de codificación independiente sin pérdidas, para la compresión reversible.

La codificación sin pérdidas no es útil para el video porque no proporciona razones de compresión altas. La codificación extendida se usa principalmente para proporcionar decodificación parcial rápida de una imagen comprimida, para que la

apariencia general de esta pueda determinarse antes de que se decodifique totalmente. Esto tampoco es útil para el video ya que éste se construye de una serie de imágenes fijas, cada una de las cuales debe decodificarse y visualizarse a un ritmo muy rápido.

De las dos alternativas de codificación de entropía, la codificación aritmética sólo se usa en los procesos de codificación sin pérdidas y extendida. Nosotros describiremos sólo el sistema básico descrito en la especificación JPEG, que usa codificación Huffman.

En el sistema básico, denominado a veces sistema básico secuencial, la precisión de los datos de entrada y de salida está limitada a 8 bits, mientras que los valores cuantificados de la DCT están limitados a 11 bits. La propia compresión se realiza en tres etapas secuenciales:

- *Cálculo de la DCT*: Se divide la imagen en bloques de píxeles de tamaño 8x8 (ver Figura 5.18), que se procesan de izquierda a derecha y de arriba abajo. Según se va encontrando cada bloque o subimagen de 8x8, se cambian los niveles de sus 64 píxeles, sustrayendo de los mismos la cantidad $2^{n-1}$, siendo $2^n$, el máximo número de niveles de gris. Esto es, para las imágenes de 8 bits se resta 128 de cada píxel. Después se calcula la Transformada Discreta del Coseno bidimensional del bloque, produciendo un conjunto de 64 valores conocidos como coeficientes de la DCT, como se ve en la Figura 5.19.

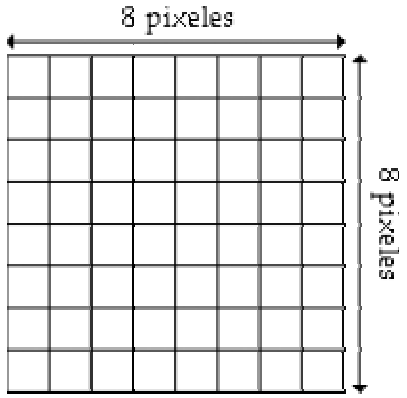



**Figura 5.18. Bloque o subimagen de tamaño 8x8.**

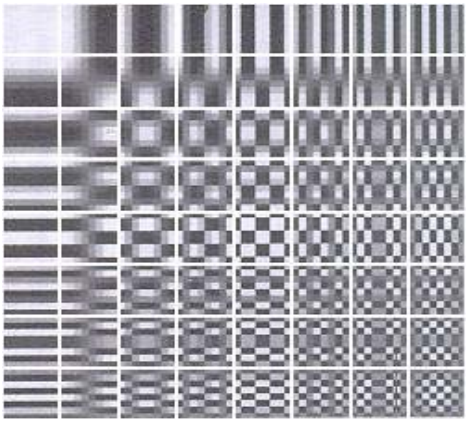


**Figura 5.19. Coeficientes de la DCT.**

- *Cuantificación de los coeficientes de la DCT:* Los 64 coeficientes son entonces cuantificados, produciendo en algunos de ellos su reducción a cero. Los coeficientes son codificados en umbral, usando una matriz de cuantificación y son preparados para la codificación de entropía convirtiéndolos en una cadena unidimensional de 64 coeficientes en orden cuasi-ascendente de los componentes de frecuencia. Para convertir los coeficientes en esta cadena unidimensional se reordenan usando una exploración o barrido en zig-zag. El primer coeficiente del barrido en zig-zag es conocido como el coeficiente DC mientras que el resto son los coeficientes AC (ver Figura 5.20). A la matriz de cuantificación se le pueden aplicar factores de escala para obtener diversos niveles de compresión. Las entradas de la matriz de cuantificación son usualmente determinadas según consideraciones psicovisuales, las cuales son discutidas más adelante.

- *Asignación del Código de Longitud Variable (VLC):* El coeficiente DC de cada bloque es codificado usando DPCM. Es decir, se codifica la diferencia entre coeficiente DC del presente bloque y el del bloque previamente codificado. Puesto que la cadena unidimensional reordenada según el barrido en zig-zag de la Figura 5.20 está distribuida cualitativamente según una frecuencia espacial creciente, el procedimiento de codificación JPEG ha sido diseñado de modo que se beneficia de la existencia de largas series de ceros que se producen normalmente en la reordenación. En particular, los coeficientes AC no nulos se codifican utilizando un código de longitud variable que define el valor del

coeficiente y el número de ceros precedentes. Se proporcionan unas tablas de especificación estándar de códigos de longitud variable.

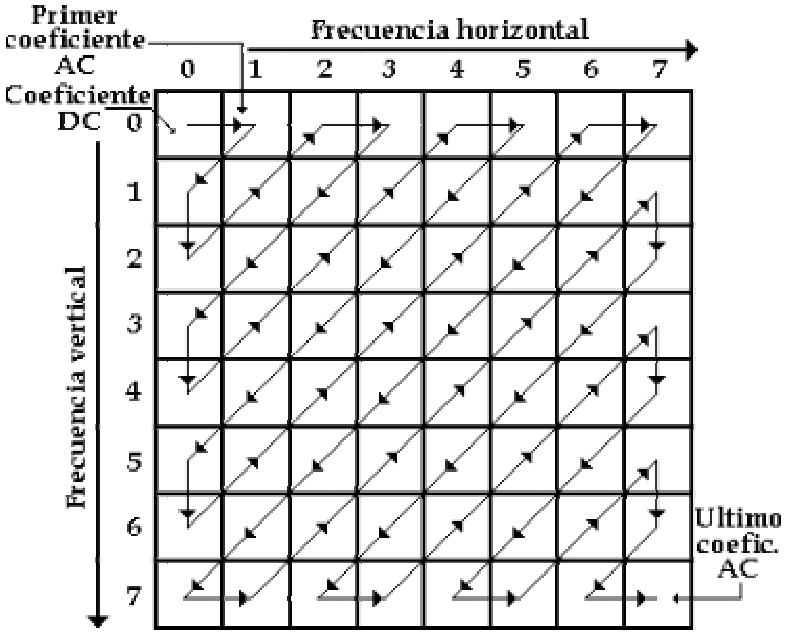



**Figura 5.20. Barrido o exploración en zig-zag.**

La Figura 5.21 es un diagrama de bloques simplificado que muestra los procedimientos involucrados en la compresión JPEG.

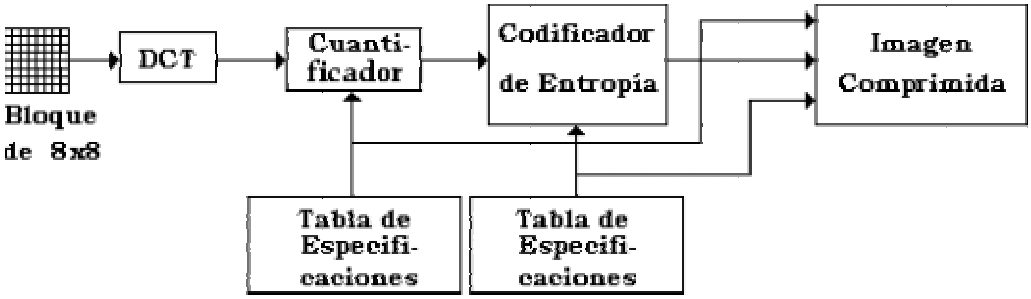



**Figura 5.21. Secuencia de procedimientos de compresión JPEG.**

La decodificación es esencialmente el proceso inverso al de la codificación. Se llevan a cabo los mismos procesos, pero en orden inverso. Las tablas de

especificación usadas en el proceso de codificación se llevan junto con el flujo de datos después de la compresión y se usan para la descompresión. El decodificador de entropía convierte el flujo de bits comprimido en una nueva tabla en zig-zag de coeficientes DCT. Estos se multiplican entonces por los coeficientes de decuantificación y se alimentan en el proceso DCT inverso IDCT (Transformada Discreta del Coseno Inversa). La salida del proceso es un bloque de píxeles reconstruido de tamaño 8x8. Por supuesto, este bloque de píxeles de 8x8 puede no reproducir exactamente el original ya que se perdió alguna información en el proceso de codificación. La Figura 5.22 es un diagrama de bloques simplificado del proceso básico involucrado en la descompresión JPEG.

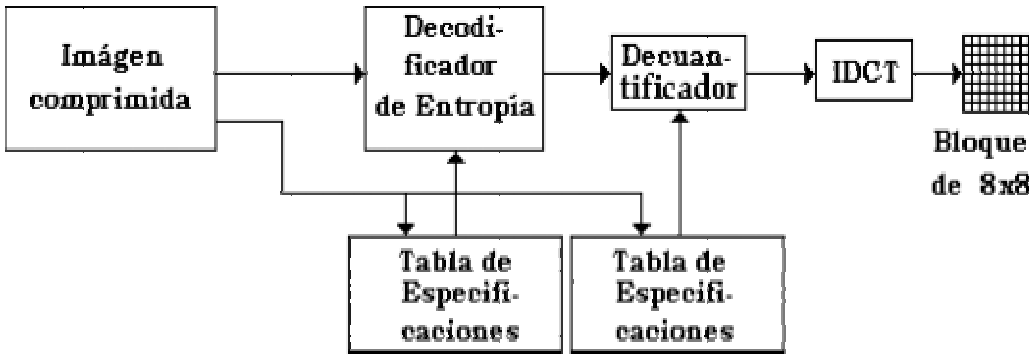



**Figura 5.22. Secuencia de procedimientos de descompresión JPEG.**

Hoy la mayoría de las imágenes electrónicas son grabadas en color, en el dominio RGB (Red, Green, Blue). JPEG transforma las imágenes RGB al espacio luminancia-crominancia, generalmente referido al dominio Y-Cr-Cb, definido como

$$Y = 0.3R + 0.6G + 0.1B$$

$$C_r = \frac{B - Y}{2} + 0.5$$

$$C_b = \frac{R - Y}{1.6} + 0.5$$

Ya que el ojo humano es relativamente insensible al contenido de altas frecuencias de los canales de crominancia Cr y Cb (ver Figura 5.24), ellos son

submuestreados por 2 en ambas direcciones. Esto es ilustrado en la Figura 5.23 donde los canales de crominancia contienen la mitad de muchas líneas y píxeles por línea comparados al canal de luminancia.

JPEG ordena los píxeles de una imagen a color como no entrelazado (3 exploraciones separadas) o entrelazado (una exploración sencilla).

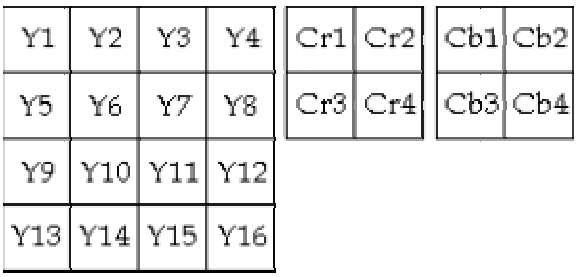



**Figura 5.23. Submuestreo de los canales de crominancia.**

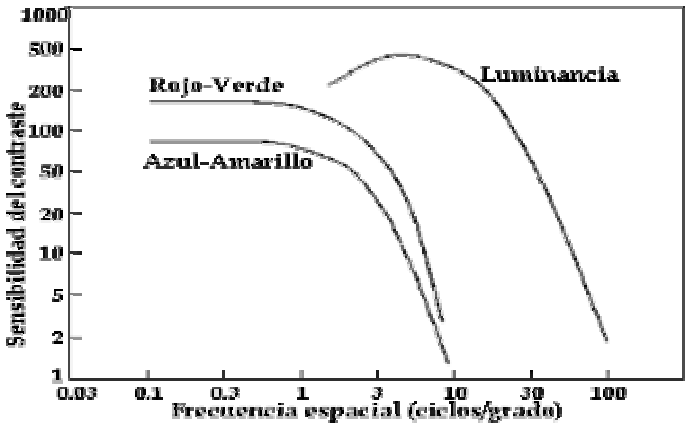



**Figura 5.24. Respuesta visual a las variaciones de luminancia y crominancia.**

El entrelazado hace posible descomprimir la imagen, y convertirla de la representación luminancia-crominancia a RGB para visualizarla con una mínima cantidad de memoria intermedia. Para los datos entrelazados, los bloques DCT son ordenados de acuerdo a los parámetros especificados en la trama.

A fin de reducir la redundancia psicovisual en las imágenes, JPEG incorpora las características del sistema visual humano en el proceso de compresión a través de la especificación de matrices de cuantificación. Se conoce que la respuesta en frecuencia del sistema visual humano decae con el incremento de la frecuencia espacial. Además, este decaimiento es más rápido en los dos canales de crominancia. La función de sensibilidad del contraste representada en la Figura 5.24 demuestra

este efecto. Esto implica que una pequeña variación en la intensidad es más visible en regiones de variación lenta que en las regiones de variación rápida, y también más visible en la luminancia comparada con una variación similar en la crominancia.

Como resultado JPEG admite la especificación de dos matrices de cuantificación, una para la luminancia y otra para los dos canales de crominancia para asignar más bits a la representación de los coeficientes que son visualmente más significativos. Las figuras 5.25 y 5.26 muestran matrices de cuantificación típicas para los canales de luminancia y crominancia respectivamente. Los elementos de estas matrices son basados en la visibilidad de funciones base individuales DCT de 8x8 con una distancia de observación igual a 6 veces el ancho de pantalla. Las funciones base fueron vistas con resolución de luminancia de 720 píxeles x 576 líneas y una resolución de crominancia de 360x576. Las matrices sugieren que estos coeficientes DCT que corresponden a imágenes base con baja visibilidad pueden ser mas toscamente cuantificados.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | 11 | 10 | 16 | 24 | 40 | 51 | 61 |
| 12 | 12 | 14 | 19 | 26 | 58 | 60 | 55 |
| 14 | 13 | 16 | 24 | 40 | 57 | 69 | 56 |
| 14 | 17 | 22 | 29 | 51 | 87 | 80 | 62 |
| 18 | 22 | 37 | 56 | 68 | 109 | 103 | 77 |
| 24 | 35 | 55 | 64 | 81 | 104 | 113 | 92 |
| 49 | 64 | 78 | 87 | 103 | 121 | 120 | 101 |
| 72 | 92 | 95 | 98 | 112 | 100 | 103 | 99 |

**Figura 5.25. Matriz de cuantificación para el canal de luminancia.**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 17 | 18 | 24 | 47 | 99 | 99 | 99 | 99 |
| 18 | 21 | 26 | 66 | 99 | 99 | 99 | 99 |
| 24 | 26 | 56 | 99 | 99 | 99 | 99 | 99 |
| 47 | 66 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
| 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |

**Figura 5.26. Matriz de cuantificación para el canal de crominancia.**

La compresión de una imagen a color sigue los siguientes pasos:

- *Descomposición en bloques o subimágenes:* La imagen original en la forma *Y*, *Cr*, *Cb* se divide en bloques de 8x8 píxeles, siendo, para una imagen en formato CCIR 601 de 720 x 576, un total de 6480 bloques de luminancia *Y* y 3240 bloques para cada una de las componentes *Cr* y *Cb*. Cada uno de estos bloques forma una matriz de 64 números de 0 a 255 (para imágenes de 8 bits) para la luminancia, y de –128 a +127 para las componentes *Cr* y *Cb* (ver la Figura 5.27).

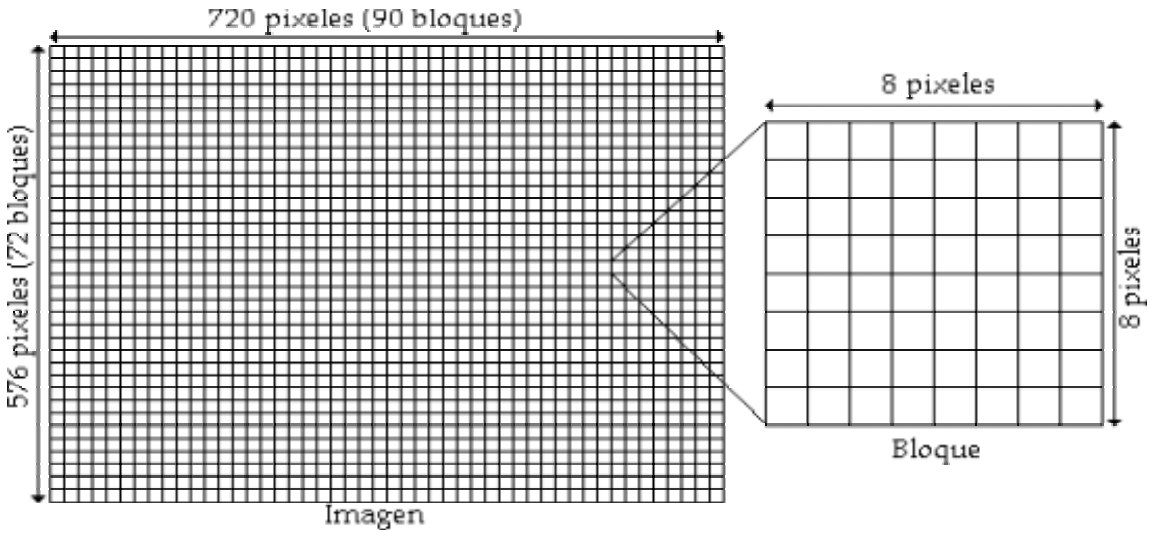


**Figura 5.27. División en bloques o subimágenes de 8x8 píxeles.**

- *Cálculo de la DCT:* Se aplica la DCT a cada uno de los bloques *Y*, *Cr*, *Cb,* generando para cada uno de ellos una nueva matriz de 8x8 compuesta por los coeficientes de las componentes de frecuencias espaciales. El valor de estos coeficientes disminuye rápidamente cuando se van alejando del origen de la matriz, terminando generalmente en una serie de ceros. De esta forma si un bloque es de luminancia y color uniformes, únicamente el primer coeficiente (coeficiente DC) no será nulo y así solo habrá que transmitir un único coeficiente en lugar de 64.

- *Discriminación por umbral y cuantificación:* Esta etapa tiene en cuenta las particularidades de la visión humana: consiste en poner a cero los coeficientes inferiores a un valor predeterminado y en codificar los coeficientes restantes con una precisión decreciente a medida que la frecuencia aumenta. El coeficiente DC se codifica en DPCM, lo que permite aumentar la precisión de cuantificación para un número de bits dado, de manera que se puede minimizar la visibilidad de los bloques sobre la imagen reconstruida, ya que el ojo, si bien es poco sensible a los detalles finos, es, por el contrario, muy sensible a pequeñas diferencias de luminancia sobre áreas uniformes.

- *Barrido en zig-zag:* Con la excepción del coeficiente DC que se trata por separado, los 63 coeficientes AC se leen en zig-zag para transformar la matriz en una cadena de datos en serie, adaptada a la próxima etapa del proceso.

- *Codificación entrópica de Huffman (VLC):* Esta última etapa consiste en codificar los coeficientes con una longitud tanto más corta cuanto más frecuentes sean estadísticamente, como se vio anteriormente.

### 5.6.3 EL ESTÁNDAR H.261

El estándar H.261 fue desarrollado (antes que MPEG) para satisfacer la compresión de video para transmisiones de bajo ancho de banda y su aplicación más extendida es la de videoconferencia [Hopper 1994]. Conocido también como px64, es considerado como un compresor del tipo lossy (con pérdida) que soporta

velocidades de transmisión con múltiplos de 64 Kbps, de ahí su gran difusión en videoconferencia sobre RDSI. Consta básicamente de cinco etapas:

- Etapa de compensación del movimiento.
- Etapa de transformación (DCT).
- Etapa de cuantificación "lossy" (con pérdidas).
- Dos etapas de codificación del tipo sin pérdidas (codificación run-length y codificación de Huffman).

H.261 es la recomendación de ITU (Unión Internacional de Telecomunicaciones, antes CCITT) para la compresión de video en sistemas de videoconferencia, siguiendo el estándar internacional H.320 [Schulzrinne et al. 1994]. Esta norma permite la utilización de anchos de banda múltiplos de 56 Kbps o 64 Kbps. H.261 tan sólo acepta dos tamaños de pantalla (CIF y QCIF) y métodos de codificación adecuados para la videoconferencia.

La digitalización proporciona grandes ventajas con respecto a la señal analógica: más robustez frente al ruido, calidad y resolución muy superiores, edición y efectos muy sofisticados, pero esta digitalización origina una cantidad de información que supera con creces las posibilidades de proceso de nuestros sistemas.

Si quisiéramos grabar una película de 3 horas, necesitaríamos una astronómica cantidad de espacio para poder almacenarla, por lo tanto debemos comprimir la información. El proceso de estandarización de los algoritmos de compresión (o como mínimo de su sintaxis) es de vital importancia y esta es la virtud de los estándares MPEG. Originalmente pensado para almacenar y reproducir señales de video digital surgió MPEG-1 con una velocidad mínima de 1.5 Mbps y calidad comparable a VHS, luego surgió MPEG-2 como evolución de MPEG-1 y pensado para la transmisión de TV digital (y audio) a través de cualquier medio (cable, satélite, terrestre) con calidad comparable a NSTC/PAL hasta HDTV, siendo elegido como estándar de transmisión por el proyecto de transmisión de TV digital DVB (Digital Video Broadcasting).

Actualmente disponemos de MPEG-4 pensado para aplicaciones multimedia con gran capacidad de interacción y MPEG-7 para búsqueda de información. Con

anterioridad, y a partir del mismo comité, surgió la especificación H.261 utilizada para videoconferencia, que se puede considerar como un subconjunto de MPEG-1.

# CAPÍTULO 6

# TARJETAS DIGITALIZADORAS

# 6 TARJETAS DIGITALIZADORAS

## 6.1 INTRODUCCIÓN

Tal y como se ha descrito en capítulos anteriores, uno de los objetivos principales de este proyecto fin de carrera consiste en transmitir desde el Servidor, según demanda por parte del Cliente, imágenes capturadas desde una cámara conectada a un PC, donde se encontraría instalado y ejecutándose dicho programa Servidor. Gracias a esta cámara y a través de una tarjeta digitalizadora, podemos digitalizar las imágenes y las secuencias de imágenes que se almacenarán en disco en forma de archivo. Recordemos que estas imágenes archivadas constituirán una base de datos de imágenes a las que el Cliente tendrá acceso en todo momento, desde cualquier máquina conectada a la red.

Por tanto, gracias a este tipo de dispositivo podemos realizar dos tareas diferentes con relación a este proyecto:

- Transmitir desde el Servidor, bajo demanda por parte del Cliente, la imagen que en ese momento está capturando la cámara.
- Almacenar en disco una imagen o una secuencia de imágenes en forma de archivo, que puede ser solicitado por el Cliente en cualquier instante del tiempo.

Por ende es de vital importancia para la persona interesada en este proyecto fin de carrera, comprender el funcionamiento de estos sistemas. De ahí que dediquemos a tal efecto una parte de este documento.

El objetivo de este capítulo, es hacer una descripción de este tipo de hardware además de introducir las nociones software necesarias para poder interpretar la interacción de los programas que forman parte de este proyecto fin de carrera, con ambas tarjetas digitalizadoras.

Es también propósito de esta parte de la memoria, que sirva a modo de manual para futuros trabajos con este hardware que se encuentra instalado y en perfecto funcionamiento en los laboratorios del citado departamento.

El equipo disponible en el laboratorio para realizar estas operaciones consiste en dos tarjetas digitalizadoras, una de la compañía Matrox y otra de la compañía Imaging Technology, junto con las cámaras descritas en capítulos anteriores. La tarjeta digitalizadora Matrox se encuentra instalada en el PC que está conectado directamente con las cámaras que están montadas en el posicionador. Por otra parte, la tarjeta de Imaging Technology está instalada en el puesto de trabajo donde se ha desarrollado este proyecto y que está en laboratorio de proyectos de la 2ª planta del Departamento de Ingeniería de Sistemas y Automática.

## 6.2 LA TARJETA DIGITALIZADORA

La función de la tarjeta digitalizadora es bastante simple. Como su propio nombre indica, este tipo de hardware permite introducir en el ordenador imágenes y sonido procedentes de una videocámara o de un reproductor de video. También permiten grabar en video, o reproducir en un monitor, la imagen de la pantalla del ordenador y su correspondiente sonido.

Además, para digitalizar de forma continua secuencias de video más largas y guardarlas luego en el disco duro, es necesario que no se corte o se interrumpa el flujo de datos. Se necesitará por tanto una CPU y un disco duro rápidos y potentes, que en conjunción con la tarjeta digitalizadora se encargarán de que la captura se realice de forma adecuada.

Algunas tarjetas también incorporan un sintonizador de TV, por lo que es posible emplear el ordenador como receptor de TV (muchas de estas tarjetas suelen incluir además un decodificador de tele-texto, mando a distancia y sintonizador de radio).

El video consume bastante espacio en la unidad de disco duro. Para capturar video se requiere que los puertos y el disco duro del computador tengan altas capacidades de transferencia de información digital (7MB por segundo o mayor). La tarjeta digitalizadora cumple, además, la función de comprimir la información recibida de la cámara, de tal forma que el video almacenado no ocupe tanto espacio en el disco duro o en otra unidad de almacenamiento del computador.

Los sistemas de compresión de video se basan en que los cuadros del video tienen mucha información redundante, se consideran únicamente las diferencias que tiene un cuadro con respecto a otro y eliminan información. Buena parte de la

información perdida no es perceptible al ojo humano. MPEG (Moving Picture Experts Group) es un estándar internacional para la compresión de video digital, es el más utilizado tanto para desarrollos multimedia como para reproducciones locales y se basa precisamente en buscar la diferencia entre imágenes y sólo registra los cambios producidos. Sobre compresión de imágenes se dedica un capítulo entero en este proyecto fin de carrera y por ello no vamos a profundizar más en este apartado.

Para la definición de la tarjeta digitalizadora hay que tener en cuenta el formato utilizado por la cámara de video: NTSC, PAL, SECAM, etc., debe existir compatibilidad entre el formato de la cámara y el de la tarjeta digitalizadora. En el sistema NTSC, por ejemplo, sólo es posible la representación de 525 líneas de imagen, así que si la cámara de video utilizada tiene más de 525 líneas, parte de la información o parte de la imagen se perderá, lo correspondiente ocurre para los demás formatos.

Si la cámara de video tiene más de 525 líneas en el formato NTSC y más de 600 en el PAL y se requiere una tarjeta digitalizadora, ésta debe ser configurable, esto es, debe permitir un ajuste entre el número de líneas de imagen de la cámara y la de la imagen almacenada, para poder utilizar toda la información que contiene la imagen capturada por la cámara de video. Lo que queremos decir es que la tarjeta debe permitir un ajuste de la frecuencia de sincronización horizontal: número de líneas que conforman la imagen, y de la vertical: frecuencia de repetición de la información que conforma la imagen.

El tipo de bus que utiliza la tarjeta es importante, porque de él, dependerá la información que pueda ser extraída de la imagen digitalizada y que pasará al PC, el bus EISA permite obtener mayor cantidad de información de la imagen que el bus ISA.

Algunas tarjetas digitalizadoras poseen una memoria y un procesador propios, aunque éste último tiende a desaparecer, debido a la alta velocidad de los microcomputadores disponibles hoy en el mercado. La memoria de la tarjeta digitalizadora depende de la cantidad de información enviada por el elemento sensor de la cámara de video (Nº. de píxeles), del software que maneje y del número de imágenes simultáneas o buffers. El tamaño mínimo de la memoria corresponde a un bit por píxel de la imagen.

Cuando la imagen se procesa en el PC, entonces la memoria dependerá del software utilizado, el tamaño de la imagen y del número de buffers.

Para poder ver en el monitor del PC la imagen que está siendo capturada por la cámara de video, "visión en vivo", se requiere un PC de prestaciones más elevadas, si no se dispone de él, es posible separar la señal producida por la cámara de video y llevar un terminal con esta señal a un T.V. y otro terminal al PC y continuar en él el resto del procesamiento.

## 6.3 LA TARJETA MATROX METEOR II

### 6.3.1 DESCRIPCIÓN HARDWARE

Como hemos mencionado anteriormente, esta tarjeta digitalizadora es la que se encuentra instalada en el ordenador del laboratorio situado en la planta baja de edificio. Esta tarjeta es la que procesa las imágenes procedentes de las cámaras que están montadas en el posicionador.

La tarjeta de la que se trata es el modelo Matrox Meteor II Multichannel y presenta las siguientes características fundamentales:

- Es una tarjeta diseñada para capturar imágenes de fuentes de video en color analógico normal o monocromas. También presenta una gran funcionalidad en sistemas de tratamiento de imágenes basados en PC's.

- Esta digitalizadora puede transferir las imágenes adquiridas bien a la memoria de sistema para procesarla o bien a la memoria de dispositivo para visualizarlas, a una tasa de video en tiempo real. Además para prevenir la pérdida de información durante los accesos al bus en sistemas con múltiples aplicaciones funcionando, esta tarjeta ofrece 4 Mbytes para almacenamiento temporal de los datos.

- Con respecto al formato de trabajo diremos que esta tarjeta puede adquirir diferentes tipos de formatos estándar de video, usando su decodificador de video. La información de la imagen puede ser procesada en tiempo real para transformarla de un formato a otro. Este tratamiento incluye sub-muestreos horizontal y vertical independientes de 2 a 16.

- Con respecto a la potencia de salida, la tarjeta Matrox Meteor II soporta cámaras alimentadas a 5 ó 12 V (DC). Esta alimentación se toma directamente de la fuente del propio ordenador, previendo la sobrecarga del bus PCI.

- La tarjeta posee un conector externo simple de 44 pines para la entrada de video, señales de sincronización y control separadas, salida DC de 5 ó 12V y un interfaz serie RS-232.

- También presenta un interfaz del Bus PCI Maestro/Esclavo de 32 bits. Cuando se configura en modo maestro, se realiza la transferencia sin ser necesaria la intervención continua de la máquina. Buffers de gran capacidad, permiten una captura de imágenes en tiempo real a la memoria de sistema bajo condiciones de sobrecarga del bus. Esta situación de sobrecarga se da en sistemas donde concurren al mismo tiempo capturas de imágenes, aplicaciones gráficas, acceso a la red, acceso a disco y operaciones de entrada-salida. Como se indicó anteriormente el interfaz PCI soporta sub-muestreo 2-16 de los datos de la imagen, con el propósito de reducir los requerimientos de ancho de banda en el bus PCI. También implementa técnicas de doble buffer, lo que permite la optimización de procesos concurrentes de adquisición y procesamiento.

- En cuanto a las especificaciones técnicas más destacadas podemos decir:
  - Entrada de video: NTCS, PAL, RS-170, CCIR.
  - Soporta VCR's.
  - Tabla de corrección de Gamma programable.
  - AGC controlable.
  - Formato RGB 8:8:8 o YUV 4:2:2.
  - Dos entradas y salidas TTL auxiliares.
  - 4 Mb SGRAM Buffer.
  - Hasta 12 entradas de video.
  - Posibilidad de disparo de la adquisición por interrupción externa.

El esquema hardware que presenta esta tarjeta es el siguiente:

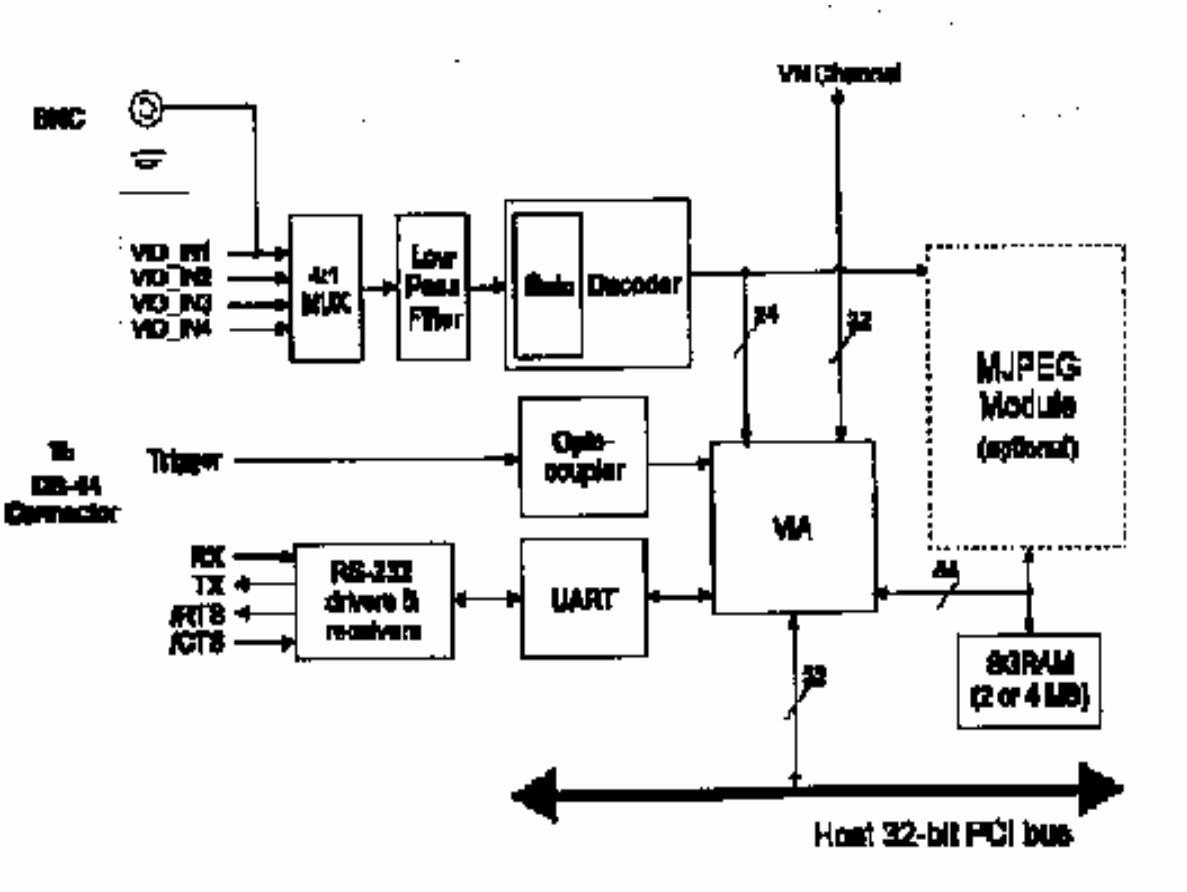



**Figura 6.1. Esquema hardware de la digitalizadora Matrox.**

En resumidas cuentas, diremos que para trabajar con el hardware presentado de la compañía Matrox necesitaremos los siguientes elementos:

- Un ordenador con un bus PCI y un procesador Intel Pentium o superior.
- Si usamos el software para tratamiento de imágenes que ofrece esta compañía necesitaremos instalar el sistema operativo Windows de Microsoft.
- El chipset del bus PCI de nuestro equipo debe ser relativamente actual, por ejemplo el 440BX, 810, 815E, 820, 840, ó 850 para aprovechar completamente la funcionalidad de la Matrox Meteor II. La recomendación sobre el chipset atiende fundamentalmente a mejorar las características de tasas de transferencia de datos.
- Se necesitará también que el ordenador posea una ranura de expansión vacía de 32 bits de longitud completa PCI.

- Como es normal se necesitará que el equipo posea una unidad de CD-ROM y un disco duro para poder instalar el software asociado a esta tarjeta.

### 6.3.2 DESCRIPCIÓN DEL SOFTWARE

La tarjeta digitalizadora Matrox Meteor II ofrece un gran abanico de posibilidades a la hora de trabajar con ella. Uno de los principales factores que hace de esta tarjeta un elemento tan versátil y a la vez tan potente es su software. Este paquete de software es accesible a través de la Biblioteca de Imagen Matrox "MIL" (Matrox Imaging Library) y sus derivadas MIL-Lite, ActiveMIL, ActiveMIL-Lite, y Matrox Inspector).

MIL es una biblioteca que proporciona una extensa lista de comandos que se usan para capturar, procesar, analizar, transferir, mostrar y archivar imágenes. Las operaciones de procesado y análisis incluyen operaciones de filtrado espacial, operaciones morfológicas, medidas, reconocimiento óptico de caracteres (OCR), reconocimiento de patrones, lectura de códigos matriciales y calibración.

MIL-Lite es un derivado de la biblioteca MIL e incluye todos los comandos de ésta para adquisición, transferencia, reproducción y archivado de imágenes.

Active-MIL es una selección de controles Active-X que están basados en MIL. Esta biblioteca derivada fue creada para su completa interacción con herramientas de desarrollo rápido de aplicaciones, como puede ser Microsoft Visual Basic o Visual C++.

Matrox Inspector es una aplicación interactiva para Windows que puede capturar imágenes, procesarlas, analizarlas y archivarlas. Los programadores que utilizan la biblioteca MIL, pueden usar la Matrox Inspector como una herramienta prototipo para agilizar el estudio de características más concretas de las imágenes sin tener que detenerse a implementar operaciones tan simples como las que lleva a cabo este pequeño interfaz.

Matrox Intellicam es un programa para Windows que permite una interacción rápida y sencilla con la cámara y permite el acceso interactivo a todas las características de captura configurables en nuestra tarjeta.

Mil permite trabajar con distintos formatos de imágenes a la hora de guardar y rescatar de disco. Los formatos contemplados son TIFF, JPEG, BMP y AVI.

También puede manipular datos tales como imágenes monocromo, almacenadas como enteros de 1, 8, 16 y 32 bits, así como formatos de 32 bits en punto flotante. Esta biblioteca también puede manipular imágenes a color almacenadas como RGB e YUV incluyendo a su vez funciones para la conversión entre formatos.

Para este proyecto fin de carrera se ha utilizado la biblioteca de funciones que ofrece la tarjeta digitalizadora para poder implementar aplicaciones en lenguaje C. De forma que el lector pueda entender de manera sencilla la parte correspondiente a la interacción con la tarjeta, entre los bloques de código que componen el proyecto y para que el mismo, en el caso en que esté interesado, pueda trabajar en sus propias aplicaciones tomando este trabajo como referencia, pasamos a describir las funciones más importantes de las que componen el software asociado a la tarjeta.

La forma de trabajar con este hardware sigue la siguiente estructura lógica:

- El primer paso es incluir en el entorno de trabajo de nuestra aplicación los ficheros de cabecera y las librerías donde quedan definidas todas las constantes y funciones utilizadas y que no son otros que:
  - mil.h
  - milsetup.h
  - mwinmil.h
  - mil.lib
  - milmet2.lib
- A continuación debemos realizar la inicialización de los parámetros con los que vamos a trabajar, estableciendo además así unas condiciones de trabajo.
- El siguiente paso consiste en realizar las operaciones correspondientes a la captura, procesado y almacenamiento en disco de las imágenes.
- Finalmente, como corresponde a la mayoría de las aplicaciones, se procede a la liberación de los espacios y entidades reservadas.

Como es preceptivo a la hora de trabajar con un lenguaje de programación como es el C, será necesario definir los tipos de datos necesarios para la identificación de cada uno de los componentes de la aplicación. La forma de realizar este paso será

definiendo los datos del tipo MIL_ID con los que trabajan las funciones de la tarjeta y entre los que encontramos:

- *MilApplication*: Identifica la aplicación en concreto.
- *MilSistem*: Identifica el sistema donde vamos a trabajar.
- *MilDisplay*: Especifica el display.
- *MilDigitizer*: Especifica la digitalizadora con la que vamos a trabajar.
- *MilImage*: Este valor es de especial importancia ya que identifica el buffer con el que va a trabajar la tarjeta y por tanto definirá la posición de memoria donde estará almacenada la imagen que se pretende procesar.

La forma de asignar los valores reales dentro del marco de la aplicación a cada uno de estos identificadores será mediante la instrucción:

*MappAllocDefault (M_SETUP, &MilApplication, &MilSystem, &MilDisplay, &MilDigitizer, &MilImage);*

Esta instrucción asocia los valores por defecto a los identificadores anteriormente definidos. En caso en que necesitemos un buffer de imagen con una características concretas, en cuanto a tamaño, profundidad de color etc. Podemos especificarlo de la siguiente forma:

*MappAllocDefault (M_SETUP, &MilApplication, &MilSystem, &MilDisplay, &MilDigitizer, &MilImage);*

*MbufAllocColor ( MilSystem, <BufSizeBand>, <BufSizeX>, <BufSizeY>, 8+M_UNSIGNED, M_IMAGE + M_GRAB + M_DISP + M_PROC, &MilImage);*

De esta manera reservamos un buffer para imagen de tamaño dado por *<BufSizeX>* y *<BufSizeY>* con una profundidad de 8 bits (unsigned char) para cada píxel y con posibilidad de grabar, mostrar o procesar en él (esto se especifica en los parámetros (*M_IMAGE + M_GRAB + M_DISP + M_PROC*). El parámetro *<BufSizeBand>* fija el tamaño de la banda del buffer, normalmente es 1L.

Para terminar con la asignación de subsistemas, es necesario establecer una conexión entre el display y el buffer de la imagen, así como entre ellos y la ventana en pantalla donde se mostrarán las imágenes. Esta operación es posible realizarla de

dos formas, dependiendo de si queremos que la imagen sea mostrada dentro de un cuadro de nuestra aplicación, en cuyo caso usaremos:

*MdispSelectWindow ( MilDisplay, MilImage, <manejador de la ventana);*

O bien si queremos que la imagen sea mostrada en una ventana propia usaremos:

*MdispSelect ( MilDisplay, MilImage);*

Otras operaciones que nos permiten modificar o configurar los displays son, por ejemplo la función configura un nuevo display:

*MdispAlloc ( Systemld, DispNum, DispFormat, InitFlag, DisplayldPtr);*

Función que libera un display.

*MdispFree ( Displayld);*

La función que nos permite hacer un zoom en el display:

*MdispZoom ( Displayld, XFactor, YFactor);*

Etc.

Uno de los parámetros que se suelen cambiar de la configuración por defecto es el canal de captura de la imagen. Los motivos de este cambio pueden ser muchos, pero en nuestro caso particular ha sido necesario por la sencilla razón de que el cable del canal por defecto no se encontraba en buen estado. La instrucción correspondiente al cambio de canal es:

*MdigChannel ( MilDigitizer, <canal>);*

Donde canal puede tomar los valores *M_CH0, M_CH1*… Entre los posibles cambios de la configuración por defecto podemos encontrar también la elección de la digitalizadora, para lo que se usa la instrucción:

*MdigControl ( Digld, ControlType, Value);*

También podemos acceder en todo momento a modificar cualquier parámetro de configuración de cada uno de los subsistemas constituyentes de la aplicación (display, digitalizadora o sistema). Para realizar este cometido se utilizan las instrucciones que a continuación se detallan:

*MdispControl ( MilDisplay, <propiedad>, <valor>);*

*MdigControl ( MilDigitizer, <propiedad>, <valor>);*

*MsysControl ( MilSystem, <propiedad>, <valor>);*

Donde propiedad indica el *<parámetro>* que se desea modificar y *<valor>* indica el nuevo valor que queremos que tome en la nueva configuración.

En caso que queramos acceder al valor actual que poseen ciertos parámetros tendremos que ejecutar las siguientes instrucciones:

MdispInquire (MilDisplay, <propiedad>, <dirección donde almacenar el resultado>);

*MdigInquire (MilDigitizer, <propiedad>, <dirección donde almacenar el resultado>);*

*MsysInquire (MilSystem, <propiedad>, <dirección donde almacenar el resultado>);*

Una vez que tenemos definidos todos los parámetros necesarios, podemos proceder a realizar las operaciones pertinentes con el equipo, así como la modificación o control de todos los parámetros que gobiernan la captura.

En lo referente a la captura de imágenes, ésta puede realizarse de forma continua o bien capturando imágenes sueltas en instantes de tiempo definidos por el usuario. Para realizar la captura continua de imágenes y a la vez mostrarlas por pantalla a modo de video real, hemos de usa la siguiente instrucción:

*MdigGrabContinuous ( MilDigitizer, MilImage);*

Podemos esperar hasta la finalización de la grabación en curso o bien pararla cuando queramos mediante las siguientes dos instrucciones respectivamente:

*MdigGrabWait( MilDigitizer, Flag);*

*MdigGrabHalt ( MilDigitizer);*

La forma de capturar una imagen en vez de capturar una secuencia de video continua es mediante la instrucción:

*MdigGrab ( MilDigitizer, MilImage);*

De esta forma el contenido de la imagen capturada se almacena en el buffer de imagen MilImage que creamos previamente en la parte de inicialización de los parámetros.

Otra de las funciones de gran utilidad que hemos utilizado en nuestro proyecto es la que nos permite volcar el contenido del buffer de imagen en otro buffer definido por el usuario (previa reserva de memoria mediante las instrucciones malloc o calloc) con el fin de realizar el procesado pertinente. Esta función es:

*MbufGet ( MilImage, <dirección del buffer>);*

También se nos permite la operación inversa, es decir, la trasferencia de los datos de una imagen desde un buffer creado y actualizado por el usuario al buffer de la imagen (MilImage). De esta forma el contenido del buffer de usuario será mostrado automáticamente por pantalla. La instrucción a utilizar será la siguiente:

*MbufPut ( MilImage, <dirección del buffer>);*

Esta tarjeta permite una opción también de gran utilidad para el desarrollo del proyecto y que permite almacenar en disco, en un archivo de formato especificado por el usuario, la imagen capturada en ese instante. La instrucción es la siguiente:

*MbufExport ( <nombre del fichero>, <formato>, MilImage);*

El formato del fichero puede ser *M_BMP, M_RAW, M_TIFF* o *M_GIF*.

Finalmente y como es habitual a la hora de implementar aplicaciones software que han de realizar una reserva de memoria, se ha de proceder a la liberación de la memoria previamente reservada. Para este cometido existen dos modos de operación: si todo el conjunto fue inicializado mediante la instrucción *MappAllocDefault*, es posible liberarlo utilizando la instrucción equivalente:

MappFreeDefault ( MilApplication, MilSystem, MilDisplay, MilDigitizer, MilImage);

El otro caso posible consiste en realizar la liberación de cada uno de los componentes del sistema paso a paso:

*MbufFree( MilImage);*

*MdispFree( MilDisplay);*

*MdigFree( MilDigitizer);*

*MsysFree( MilSystem);*

*MappFree( MilApplication);*

En este punto tenemos que decir que la biblioteca de funciones MIL posee muchas más instrucciones de las aquí descritas pero que no hemos incluido por los motivos siguientes:

- La funcionalidad de estas es muy específica y sólo resultan útiles en aplicaciones muy concretas.
- Las instrucciones que se han descrito son suficientes para el desarrollo de aplicaciones que hagan un uso normal de la tarjeta dado que cubren la mayoría de la funcionalidad que ofrece.
- Para entender la parte relativa a las tarjetas digitalizadoras que encontramos en este proyecto es más que suficiente con la parte que hemos descrito anteriormente.
- Remitimos en el caso en que el usuario necesite más información sobre el software, a la biblioteca de funciones Mil que podrá encontrar en la página web del fabricante cuya dirección podrá encontrar en la parte de Bibliografía de este documento.

## 6.4 TARJETA PC-COMP DE IMAGING TECHNOLOGY

### 6.4.1 DESCRIPCIÓN DEL HARDWARE

Como se ha descrito a lo largo del presente documento, la tarjeta PC-COMP de la compañía Imaging Technology se encuentra instalada en el equipo situado en el puesto de trabajo donde se ha desarrollado este proyecto fin de carrera. Este puesto se encuentra en el laboratorio de Proyectos de la segunda planta del edificio L1 de los Talleres y Laboratorios de la Escuela de Ingenieros de Sevilla.

Centrándonos en la parte hardware podemos decir en lo referente al interfaz con la cámara que esta tarjeta presenta un preciso convertidor analógico-digital (ADC), un PLL (Phase Locked-Loop) y un interfaz a la memoria de imagen de la PC-COMP. El decodificador NTSC/PAL, ganancia, offset y circuitería asociada al acondicionamiento de señal se encuentran en la interfaz de la tarjeta con la cámara.

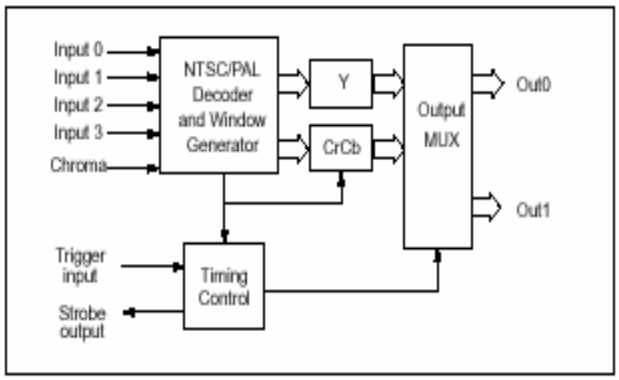



**Figura 6.2. Esquema hardware del interfaz con la cámara.**

En cuanto a la memoria de imagen hemos de decir que PC-COMP posee una memoria de 1 Mb para almacenar la imagen en la transferencia de datos entre la cámara y el sistema. Las imágenes son adquiridas y almacenadas en la memoria de imagen a una velocidad menor a los 20 Mhz y son transferidas a una velocidad superior a los 60 Mhz a la memoria RAM del equipo, bien para ser procesada o bien al dispositivo de visualización (Tarjeta de video). Esta memoria de imagen de la tarjeta garantiza la integridad de los datos durante la transmisión a la memoria del sistema. Dicha integridad podría verse afectada por los procesos de acceso al bus PCI. Esta memoria permite igualmente la adquisición de múltiples imágenes para reducir la sobrecarga del sistema.

En lo referente al interfaz con el bus PCI diremos que este permite a la PC-COMP funcionar como maestro en el bus; escribiendo así los datos de imagen directamente en otro dispositivo PCI como puede ser la memoria de sistema o la memoria de la tarjeta de video. Tasas de transferencia superiores a los 60 Mb/s pueden ser soportadas dependiendo de la capacidad del propio sistema. Las imágenes pueden ser transferidas a la memoria del sistema en una fracción del tiempo en el que fueron adquiridas. De esta forma, minimizando la tasa de transferencia, la mayoría del ancho de banda permanece disponible para el uso de este por otras aplicaciones. Este interfaz con el bus PCI que presenta la tarjeta también permite a los dispositivos conectados a este bus, acceder a la memoria de imagen de la tarjeta y a los registros que se encuentran en esta, así como interrupciones y registros de configuración tal y como requieren las especificaciones del bus PCI. PC-COMP proporciona una serie de fuentes de interrupción como son

la adquisición de la imagen, errores del bus PCI y finalización de transferencias como maestro en dicho bus.

Tal y como se puede apreciar en el esquema que se muestra en la figura 6.3 la arquitectura se podría dividir en dos partes fundamentales: la encargada de la comunicación con el bus PCI y la encargada de la adquisición de las imágenes.

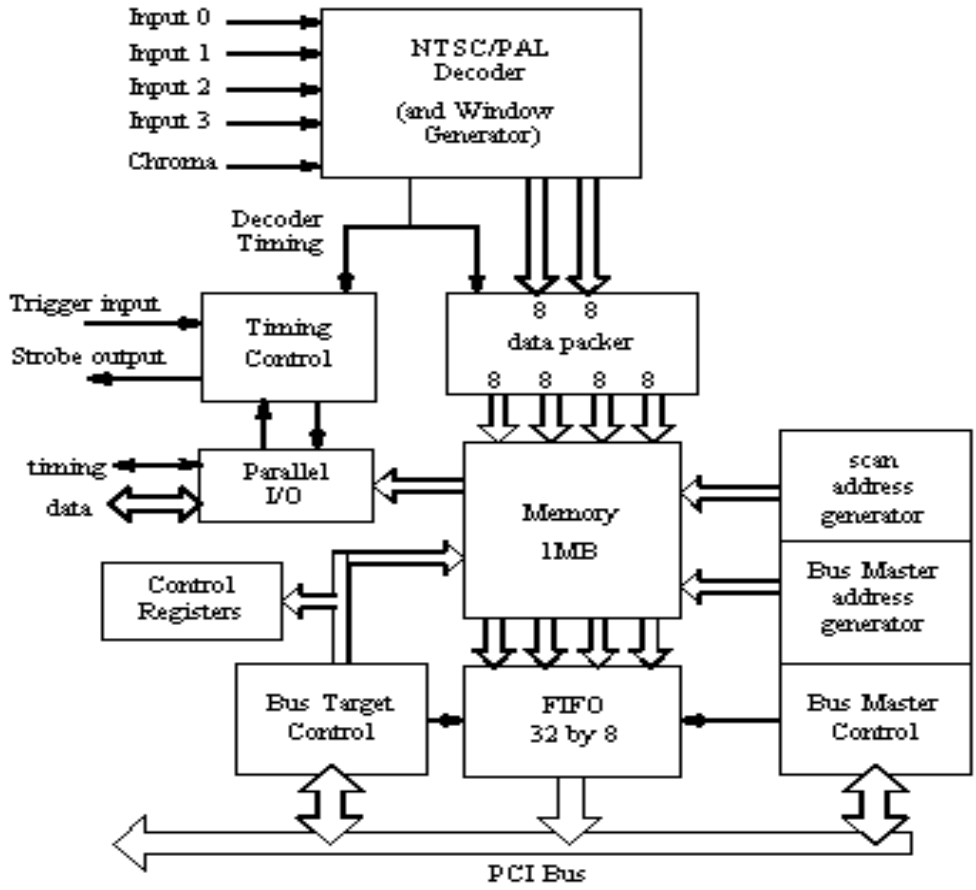



**Figura 6.3. Esquema hardware de la tarjeta PC-COMP.**

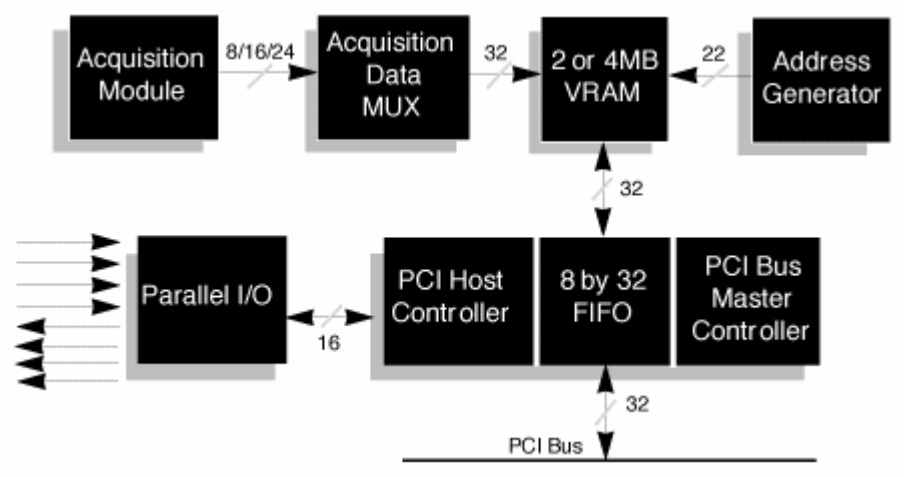



**Figura 6.4. Arquitectura del módulo de comunicacion con el bus PCI.**

Las especificaciones técnicas que presenta la tarjeta son las siguientes:

- Digitalización de señales Monocromo RS-170 y CCIR.
- Digitalización de señales en Color NTSC, PAL y S-Video.
- Cuatro entradas del video analógicas AC pareadas e impedancia de terminación de 75 Ω.
- Posibilidad de adquisición programable por evento externo a través de trigger. El ancho mínimo del pulso de entrada es de 50 nanosegundos. La polaridad también es programable.
- Salida de Strobe. Programable para pulso negativo o positivo.
- Entrada/Salida digital de 8 bits para la entrada y para la salida.
- Entradas de interrupción, dos de tipo TTL en los conectores de video (MISC) y otra también de tipo TTL en el puerto de entrada/salida digital.
- Trasferencia en modo maestro sobre el bus PCI de alta velocidad (superior a los 60 Mb/s).
- Posibilidad de adquisición y lectura por parte del sistema simultáneas.
- Resolución programable para todos los modos de trabajo.

- Profundidad en bits:
  - 8 bits para RS-170 y CCIR.
  - 16 bits para YCrCb (4:2:2) para NTSC, PAL y S-VHS.
- Tasa de captura:
  - 30 frames por segundo para RS-170, NTSC y PAL.
  - 25 frames por segundo para CCIR, PAL y S-VHS.
- Características adicionales:
  - Filtros paso de baja antialiasing y Croma&Luma.
  - Brillo, contraste y saturación programables.
  - Escalado por hardware de la imagen.
  - Respuesta en 4 milisegundos por imagen.
  - Posibilidad de uso de doble buffer.

La tarjeta PC-COMP posee un conector BNC para una entrada de disparo externo mediante una fuente de +5 Voltios. Cuando se conecta a esta entrada de trigger un sensor industrial típico, es recomendable usar un aislante óptico ya que esta entrada es bastante sensible al ruido.

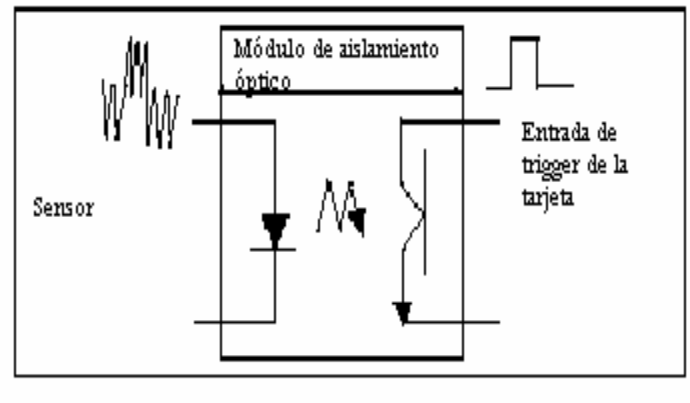



**Figura 6.5. Esquema de la entrada de trigger.**

Esta tarjeta digitalizadora presenta también una salida de Strobe que puede generar un pulso de 5 voltios.

En cuanto al resto de entradas-salidas diremos que este dispositivo posee 8 entradas y 8 salidas digitales de 16 bits de tipo TTL. Los puertos configurados como entradas o salidas no puedes intercambiarse. Las características que ofrecen estos puertos son las siguientes:

- Entrada de generación de interrupción.
- Bloqueo de la entrada.
- Lectura del estado de los datos bloqueados o de los estados actuales en los pines de entrada.
- Salida de strobe para dispositivo externo de bloqueo.

La tarjeta PC-COMP acepta datos provenientes de los siguientes formatos de video:

- RS170 (mono): 512x480, 640x480, 720x480.
- CCIR (mono): 512x512, 720x576, 768x576.
- NTCS (color): 640x480, 720x480.
- PAL (color): 765x576, 720x576.
- S-Video (color): 640x480.

Esta tarjeta digitaliza fuentes de video monocromas compuestas , como las RS-170 ó CCIR en datos de 8 ó 16 bits monocromos. Además digitaliza señales compuestas en color cuyo origen son los formatos NTCS o PAL en formato de datos de 16 bits 4:2:2 YCrCb.

Todas las cámaras que cumplan con las especificaciones descritas en la lista anterior con resoluciones estándar, son fuentes de video aceptables. Asimismo la tarjeta PC-COMP solamente acepta fuentes de video compuestas. Las fuentes de video compuestas son aquellas donde las señales de tiempo están integradas en una señal simple, como RS-170, CCIR, NTCS y PAL. Un circuito PLL (que posee la tarjeta internamente) se encarga de la sincronización de las señales de las fuentes de video compuestas. Esta tarjeta no soporta cámaras que proporcionen temporizaciones separadas. Cámaras de barrido variable pertenecen a esta categoría.

Existe compatibilidad con salidas de video desde VCR o de un televisor. Las siguientes fuentes de video no las soporta esta tarjeta:

- Cámaras de barrido variable.
- Cámaras de barrido progresivo. Cámaras sin entrelazado.
- Cámaras RGB. Cámaras en color que proporcionan información del color Rojo, Verde y Azul en señales a través de líneas separadas.
- Cámaras configuradas para aceptar señales de sincronización de una capturadora de imagen. PC-COMP no puede sacar ninguna señal de sincronización de video.

### 6.4.2 DESCRIPCIÓN DEL SOFTWARE

Tal y como hemos hecho anteriormente para el caso de la tarjeta de la compañía Matrox, procedemos a introducir la forma de emplear la funcionalidad de la tarjeta de Imaging Technolgy mediante la potente herramienta Software que incorpora este dispositivo. Describiremos en primer lugar el procedimiento a seguir a la hora de desarrollar una aplicación que utilice esta tarjeta para después continuar detallando, dentro de la amplia gama de funciones, aquellas que consideramos más importantes para el desarrollo de aplicaciones en lenguaje C que pretendan incorporar este tipo de Hardware.

Como sucede con la mayoría del software que los fabricantes de hardware suministran para el uso de sus productos en aplicaciones generadas por los usuarios, este se hace accesible a los programadores a través de librerías. Las librerías que proporciona el fabricante para el manejo de esta tarjeta son tres:

- Librería especifica para al módulo de adquisición AM-STD-COMP (ams.h), la cual da acceso a aspectos relativos a la adquisición de imágenes (canal por el que se captura, multiplexor de salida...).
- Librería para el manejo de los aspectos relacionados con la adquisición pero a nivel de estructuras de datos en memoria (puntero a la tarjeta, numero de tarjetas instaladas...). Con esta librería conseguimos un nivel de abstracción en el cual todos los aspectos referentes al módulo de captura son transparentes para el usuario. Esta librería puede trabajar con todas las tarjetas IC-PCI independientemente del modulo de adquisición que posean. Las cabeceras de las funciones están en al archivo icp.h. Todas las funciones pertenecientes a esta librería llevan el prefijo icp_. Para el correcto funcionamiento de la librería

necesita que también estén disponibles las funciones de la tercera librería lo que se consigue incluyendo en la aplicación el archivo de cabecera itxcore.h, que contiene las definiciones de las funciones incluidas en esta tercera librería.

- Por último una librería que proporciona un nivel de abstracción mayor (volcado imágenes a ficheros). Sin duda esta es la librería más importante de las tres en ella encontramos casi toda la funcionalidad necesaria para el manejo de la tarjeta, además algunas de las funciones de la librería anteriormente descrita aparecen de nuevo en esta haciendo así el código más portable a otras tarjetas. Esta librería crea una serie de estructuras para realizar los accesos a la tarjeta, estas estructuras guardan información acerca de las tarjetas instaladas así como del módulo de adquisición del que se dispone, que siempre corresponde al slot0, dentro del array de tarjetas disponibles (solo es posible tener un modulo de adquisición por tarjeta). Como en el caso anterior todas las funciones devuelven un cogido para saber si se ha producido algún error y cual durante la ejecución de la función. Las funciones de esta librería empiezan por *itx_* . Esta librería crea automáticamente una ventana para la visualización de los errores y los mensajes que se estimen necesarios dentro de nuestra aplicación, para ver estos mensajes se usa la función *itx_display*.

La librería con más utilidad es la tercera, aunque no se puede prescindir de las otras dos ya que algunos aspectos son demasiado específicos como para que una librería de "alto nivel" pueda acceder a ellos.

La forma de trabajar con esta tarjeta digitalizadora consiste en seguir sistemáticamente los siguientes pasos:

- Inicialización de la tarjeta.
- Captura de imágenes.
- Finalización de uso de la tarjeta.

Describiremos a continuación las funciones más importantes que nos proporciona el Software de este dispositivo, pero hemos de tener siempre presente que dichas funciones se emplean para llevar a cabo las fases enumeradas anteriormente.

Para realizar la inicialización de la tarjeta se emplea la función:

*itx_initialize(*pSTRING fichero_de_configuracion,

Short nivel_de_error,

PSTRING nombre_de_la_tarjeta,

short *numero_de_la_tarjeta,

pWORD *ancho, pWORD *alto,

ITX_DEPTH *numero_de_bits,

ITX_COLOR *numero_colores*);*

Inicializa la tarjeta y devuelve un puntero a la estructura en memoria. Se le pasa el nombre de un fichero que contiene la configuración correcta. Este fichero contiene información como: el tipo de cámara, si es color o blanco y negro, el tamaño de la imagen, numero de píxeles, etc. Este archivo de inicialización puede ser generado a partir de una sencilla aplicación que suministra el fabricante la cual nos permite a través de un sencillo interfaz, programar los parámetros necesarios para el correcto funcionamiento de la tarjeta. Los parámetros que toma esta función son:

*nivel_de_error*: nivel de error a usar:

ABORT - Si se produce algún error se para la ejecución de la aplicación

WARNING - Sólo se informa del error (por defecto).

INQUIRE - Devuelve el nivel de error actual.

*nombre_de_la_tarjeta*: tipo de la tarjeta de la que se dispone

"PCCOMP" si es PC-COMP (esta, en nuestro caso)

"ICA" si es IC-ASYNC

"ICP" si es IC_PCI

"NULL" si es Pcvision.

*numero_de_la_tarjeta*: numero de la tarjeta, si hay más de una, valores entre 0 y 3.

ancho: devuelve el ancho con el que esta configurada la tarjeta.

alto: devuelve el alto con el que esta configurada la tarjeta.

*numero_de_bits*: devuelve el numero de bits por pixel

ITX_PIX8 - 8 bits por pixel.

ITX_PIX16 - 16 bits por pixel.

*numero_colores*: devuelve el tipo de captura (color o escala de grises)

ITX_MONO - imágenes monocromas (8bits)

ITX_YCRCB - imágenes en pseudocolor (16 bits)

La función devuelve NULL si no ha sido posible inicializar la tarjeta.

*itx_set_port_camera (pMODCNF mod, pSTRING nombre_camara, short puerto)*; Esta función selecciona el puerto de entrada, es decir, el canal por el cual se realiza la captura. Los parámetros que se le han de pasar a esta función son respectivamente el puntero a la estructura en memoria, el nombre de la cámara a usar (debe ser uno de los nombres definidos en el fichero de configuración) y el número de puerto de captura (debe ser un valor entre 0 y 3).

En cuanto al proceso de adquisición de imágenes, podemos agrupar el siguiente conjunto de funciones:

*itx_grab (pcmod, ITX_CAM_FRAME)*; Esta función activa la captura continua de imágenes. Los parámetros que toma esta función son el puntero a la estructura en memoria devuelto por itx_initialize y el identificador del frame en memoria.

*itx_freeze(pcmod, ITX_CAM_FRAME)*; Función que detiene la captura continua. Los parámetros que toma esta función son los mismos que los de la función anterior.

*itx_snap(pcmod, ITX_CAM_FRAME)*; Función que realiza la captura de una imagen. Los parámetros son los mismos que los de las dos funciones anteriores.

*itx_host_snap( pMODCNF mod, BYTE * dest)*; Realiza la captura de una imagen y la guarda en una posición de memoria. Como primer parámetro toma el puntero a la estructura en memoria devuelto por itx_initialize y como segundo parámetro se le pasa el puntero a la dirección de memoria donde se va a guardar la captura. Este puntero deberá apuntar a una zona de memoria previamente reservada con el tamaño adecuado.

*itx_display_frame(pcmod, ITX_CAM_FRAME, ITX_SNAP)*; Visualiza la captura en el programa PCIview.exe que debe ser cargado con la función de Windows

winexec. El tercer parámetro puede ser sustituido por ITX_GRAB si lo que se pretende es visualizar la captura continua de imágenes.

*itx_ycrcb_to_rgb ( BYTE * dest, BYTE * src, DWORD numero_bytes, int modo)*; Función que realiza la conversión de una imagen de formato YCRCB a RGB. Los parámetros que toma esta función son respectivamente un puntero al buffer destino (debe ser de tamaño suficiente para contener la imagen RGB), puntero al buffer con la imagen origen, número de bytes a convertir (tamaño en bytes de la imagen origen) y el tipo de conversión. Este último parámetro podrá tomar los siguientes valores:

ITX_YCRCTORGB16 – dest y src son del mismo tamaño.

ITX_YCRCTORGB24 – dest debe tener 1.5 veces el tamaño de src.

ITX_YCRCTORGB32 – dest debe tener un tamaño dos veces superior a src.

*itx_write_image_file( pSTRING nombre_fichero, BYTE * src, short x, short y, short dx, short dy, short clase, short bits, short compresión)*; Esta función escribe una imagen en un archivo en formato .tiff. Como parámetros se le han de pasar en este orden el nombre del fichero, un puntero que apunte al buffer con la imagen, desplazamiento en x de inicio de la imagen (normalmente 0), desplazamiento en y del inicio de la imagen (normalmente 0), ancho de la imagen en píxeles, alto de la imagen en píxeles, clase (IFFCL_GRAY o IFFCL_RGB), bits que debe vales 8 y finalmente el tipo de compresión (normalmente IFFCOMP_NONE).

Entre otras funciones de uso menos frecuente destacaremos las siguientes:

- Funciones de configuración e inicialización:
  - *itx_exit*: Salimos de la aplicación.
  - *itx_init*: Inicializa la tarjeta.
  - *itx_init_sys*: Inicializa el sistema de adquisición.
  - *itx_load_cnf*: Carga la configuración desde un fichero de configuración.
  - *itx_remove_sys*: Cierra el sistema de adquisición y libera memoria.
  - *itx_display*: Muestra mensajes en la ventana de mensajes.
- Funciones para el manejo de imágenes y ficheros:
  - *itx_read_image*: Lee una imagen desde un fichero.

- *itx_write_image*: Escribe una imagen en un fichero.

- Funciones para el manejo y procesado de la captura (envían la imagen a un frame en memoria):
  - *itx_read_area*: Lee una determinada área de un frame en memoria.
  - *itx_wait_acq*: Espera a que se termine la captura.
  - *itx_write_area*: Escribe una determinada área en un frame en memoria.
- Funciones para el manejo y procesado de la captura (envían la imagen a un buffer):
  - *itx_host_grab*: Activa la captura continua.
  - *itx_host_grab_stop*: Detiene la captura continua.
  - *itx_host_snap*: Realiza la captura de una imagen.
  - *itx_host_seq_snap*: Realiza la captura de unas cuantas imágenes.
- Funciones de alto nivel:
  - *itx_get_acq_dim*: Obtiene las dimensiones de la captura.
- Funciones para mostrar imágenes: Estas funciones sirven para crear una ruta entre la fuente de la imagen, dispositivo al que se le manda y la propia conexión entre ambas.
  - *itx_create_img_conn*: Crea la estructura de conexión.
  - *itx_create_img_server*: Crea un servidor de imágenes.
  - *itx_delete_img_conn*: Borra la estructura de conexión.
  - *itx_delete_img_server*: Borra el servidor de imágenes.
  - *itx_delete_img_sink*: Borra la estructura de destino de la imagen.
  - *itx_delete_img_src*: Borra la estructura de la fuente de la imagen.
  - *itx_resume_img_server*: Reactiva el servidor de imágenes.
  - *itx_suspend_img_server*: Suspende el servidor de imágenes.
  - *itx_create_frame_img_src*: Crea la estructura para la fuente de imágenes desde un frame.

- *itx_create_host_img_src*: Crea la estructura para la fuente de imágenes desde un buffer.

Remitimos al lector a la bibliografía de este documento en caso que quiera ampliar la información aquí suministrada sobre el manejo del software de esta tarjeta digitalizadora.

## 6.5 CÁMARA JAI 2060 TWIN

La cámara CCD empleada para la realización de este proyecto fin de carrera ha sido el modelo 2060 Twin de la compañía JAI.

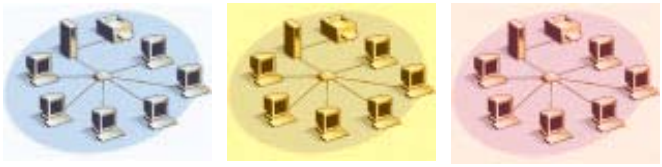
**Figura 6.6. Cámara JAI 2600 TWIN.**

Se ha empleado este dispositivo para realizar los ensayos durante el desarrollo del proyecto. Sus características principales son:

- La cámara funciona con una alimentación de continua de 12V.
- Permite alterna el funcionamiento tanto en color como en blanco y negro mediante un botón de selección que incorpora la cámara en su parte trasera.
- Este dispositivo posee sensores de tipo Hyper HAD (Hole Accumulation Diode). La corriente de oscuridad se minimiza con el uso de este tipo de dispositivo.
- Alta resolución. 450 líneas de televisión en horizontal. Salidas de video VBS y Y/C.

- Alta sensibilidad por debajo de 0.2 lux para una excelente calidad del color y por debajo de 0.002 lux en el modo blanco y negro.
- Obtiene imágenes de gran nitidez gracias a los circuitos de corrección de apertura horizontal y vertical que incorpora la cámara.
- No introduce distorsión geométrica.
- Presenta una gran inmunidad a vibraciones y golpes.
- Puede funcionar tanto con PAL (color) como con CCIR (b/n).
- El sistema de barrido es entrelazado 2:1.
- Realiza equilibrado de blanco de forma automática o manual.
- La ganancia máxima en color es de 16 db mientras que en blanco y negro es de 24 db.

Para ampliar la información referente a esta cámara remitimos al lector a la página web del fabricante que la podrá encontrar en la bibliografía de este proyecto.



# CAPÍTULO 7

# DESARROLLO DE LA APLICACIÓN CLIENTE

# 7 DESARROLLO DE LA APLICACIÓN CLIENTE

## 7.1 OBJETIVOS DEL CAPÍTULO

Procederemos a lo largo de este capítulo a detallar el funcionamiento de la primera de las aplicaciones desarrolladas en este proyecto fin de carrera: la aplicación Cliente.

En primer lugar se hará una introducción a las clases y a los objetos de C++, con el fin de que el lector menos especializado en la materia pueda llegar a comprender la filosofía de este lenguaje de programación. A continuación, pasaremos a describir las clases y su estructura jerárquica dentro del programa, detallando las funciones más destacadas de cada una de ellas y los atributos más determinantes en el funcionamiento de los programas. Finalmente se procederá a describir el interfaz gráfico que ha sido diseñado para esta aplicación.

Con el fin de facilitar la comprensión del código por parte del lector, a lo largo del capítulo se describirán las aplicaciones que no incluyen compresión de imágenes a la hora de transmitir. De esta forma se simplifica de manera considerable la descripción de los programas. La decisión que nos ha llevado a hacerlo de esta manera es que la compresión de imágenes se considera a lo largo del proyecto como una opción que se propone al usuario de estas aplicaciones, ya que bajo determinadas circunstancias esta compresión pudiera resultar innecesaria, tal y como se explica en el capítulo 2**.**

Otro de los motivos que nos ha llevado a proceder de esta manera es que gracias al lenguaje de programación tan estructurado que se ha empleado para trabajar, hemos compactado todas las funciones correspondientes a la compresión en una sola clase (CCompresion). Esta clase se añade a los programas implementados sin compresión y las modificaciones que hay que realizar en estos son prácticamente insignificantes, por lo que la mayoría del código es exactamente igual en ambos casos. Por ello y con el fin de no repetir explicaciones innecesarias, cuando se proceda a explicar la parte del código referente a la compresión, nos limitaremos a detallar la clase correspondiente y la forma en que se han modificado las aplicaciones base para el correcto funcionamiento del conjunto.

De igual forma, puesto que este proyecto fin de carrera ha sido implementado para funcionar con dos tarjetas digitalizadoras de diferentes fabricantes y por tanto con diferentes características, se han desarrollado los programas Cliente y Servidor tanto para la tarjeta de la compañía Imaging Technology como para la tarjeta Matrox. De nuevo reseñamos que gracias a la estructura de los códigos que nos permite generar Visual C++, las funciones que trabajan con la tarjeta así como las características configurables de cada una se han agrupado en una clase (CDigitizer) con el fin de que el resto del código se altere lo menos posible a la hora de trabajar con una o con otra. Esto implica que salvo esta clase anteriormente mencionada, el resto de código permanece igual, exceptuando pequeñas modificaciones. Por tanto es preferible centrarnos en detallar los programas diseñados para trabajar con una de las tarjetas (en este la de la compañía Imaging Technology) y cuando proceda entonces detallar la clase correspondiente a la otra digitalizadora junto con las modificaciones necesarias en el resto del programa para que la aplicación funcione correctamente.

Una vez explicado todo esto y a modo de resumen diremos que las aplicaciones que se detallan en este y en el siguiente capítulo corresponden a las aplicaciones Cliente y Servidor sin compresión y para la tarjeta PC-COMP de Imaging Technology. Cuando existan distintas versiones de una clase se expondrán todas ellas y se explicarán las modificaciones que en las demás se han de realizar para que el sistema completo funcione.

## 7.2 ESTRUCTURAS Y CLASES

Una clase es un tipo definido por el usuario que describe los atributos y métodos que definen las características comunes a todos los objetos de esa clase. Un objeto es una unidad que engloba en sí mismo datos y procedimientos necesarios para el tratamiento de esos datos. En C++, los atributos reciben el nombre de datos miembros y los métodos el nombre de funciones miembro de la clase. Los datos miembros nos proporcionan información sobre un determinado objeto y las funciones miembro son las operaciones que definen su comportamiento. Forman parte de estas funciones los constructores y destructores, que permiten crear e inicializar un nuevo objeto y destruirlo respectivamente.

El concepto de clase incluye la idea de ocultación de datos. Esta ocultación consiste básicamente en restringir el acceso directo a los datos. El acceso a los datos miembro de una clase solamente se permite a través de las funciones miembro públicas de esa clase. Esto indica que el usuario tendrá acceso únicamente a un conjunto de funciones públicas que permitirán la modificación de las cualidades de un determinado objeto, aún ignorando la disposición de estos atributos. De esta forma se logran dos objetivos importantes:

- Que el usuario no tenga acceso directo a la estructura interna de la clase, con el fin de que no pueda generar código basado en la estructura de datos.
- Que si en un momento determinado alteramos la definición de la clase, excepto el prototipo de las funciones miembro, todo el código escrito por el usuario no tenga que ser modificado.

Hemos de considerar que si el primer objetivo no se cumpliera, cuando se diera el segundo objetivo el usuario tendría que rescribir el código que hubiera desarrollado basándose en la estructura interna de los datos.

La característica más importante de C++ es su soporte para implementar clases. De esta forma, mientras que un ejemplar de un tipo predefinido lo hemos denominado siempre *variable*, un ejemplar de una clase se denominará *objeto*. Por otra parte, si la programación estructurada se interesaba primero por los procedimientos y después por los datos, el diseño orientado objetos se interesa en primer lugar por los datos. Para obtener más información acerca de las características de la programación orientada a objetos remitimos al lector al capítulo 2 (apartado 2.6.1) de este documento.

Una de las principales propiedades de las clases es la *herencia*. Esta propiedad nos permite crear nuevas clases a partir de clases existentes, conservando las propiedades de la clase original y añadiendo otras nuevas.

La nueva clase obtenida se conoce como *clase derivada*, y las clases a partir de las cuales se deriva, *clases base*. Además, cada clase derivada puede usarse como clase base para obtener una nueva clase derivada. Y cada clase derivada puede serlo de una o más clases base. En este último caso hablaremos de *derivación múltiple*.

Esto nos permite crear una jerarquía de clases tan compleja como sea necesario.

La derivación de clases es el principio de la programación orientada a objetos. Esta propiedad nos permite encapsular diferentes partes de cualquier objeto real o imaginario, y vincularlo con objetos más elaborados del mismo tipo básico, que heredarán todas sus características.

Las jerarquías de clases se usan especialmente en la resolución de problemas complejos, es difícil que se tenga que recurrir a ellas para resolver problemas sencillos.

Los especificadores de acceso a las clases base definen los posibles tipos de derivación: *public*, *protected* y *private*. El tipo de acceso a la clase base especifica cómo recibirá la clase derivada a los miembros de la clase base. Si no se especifica un acceso a la clase base, C++ supone que su tipo de herencia es privado.

- *Derivación pública (public).* Todos los miembros public y protected de la clase base son accesibles en la clase derivada, mientras que los miembros private de la clase base son siempre inaccesibles en la clase derivada.

- *Derivación privada (private).* Todos los miembros de la clase base se comportan como miembros privados de la clase derivada. Esto significa que los miembros public y protected de la clase base no son accesibles más que por las funciones miembro de la clase derivada. Los miembros privados de la clase siguen siendo inaccesibles desde la clase derivada.

- *Derivación protegida (protected).* Todos los miembros public y protected de la clase base se comportan como miembros protected de la clase derivada. Estos miembros no son, pues, accesibles al programa exterior, pero las clases que se deriven a continuación podrán acceder normalmente a estos miembros (datos o funciones).

Hemos creído conveniente introducir en este punto el concepto de constructor y destructor de una clase, porque son conceptos que se van a emplear en a lo largo de este capítulo.

Un *constructor* es una función especial que sirve para construir o inicializar objetos. En C++ la inicialización de objetos no se puede realizar en el momento en que son declarados; sin embargo, tiene una característica muy importante y es disponer de una función llamada constructor que permite inicializar objetos en el

momento en que se crean. En definitiva decimos que un constructor es una función que sirve para construir un nuevo objeto y asignar valores a sus datos miembros. Se caracteriza por:

- Tener el mismo nombre de la clase que inicializa.
- Puede definirse inline o fuera de la declaración de la clase.
- No devuelve valores.
- Puede admitir parámetros como cualquier otra función.
- Puede existir más de un constructor, e incluso no existir. Si no se define ningún constructor de una clase, el compilador generará un constructor por defecto que no contiene argumentos.

Un *destructor* es una función miembro con igual nombre que la clase, pero precedido por el carácter ~. Una clase sólo tiene una función destructor que, no tiene argumentos y no devuelve ningún tipo. Un destructor realiza la operación opuesta de un constructor, limpiando el almacenamiento asignado a los objetos cuando se crean. C++ permite sólo un destructor por clase. El compilador llama automáticamente a un destructor del objeto cuando el objeto sale fuera del ámbito. Si un destructor no se define en una clase, se creará por defecto uno que no hace nada. Normalmente los destructores se declaran public.

Una vez introducidos estos conceptos básicos de la programación orientada a objetos consideramos que el lector menos experimentado estará preparado para poder entender lo que aquí se propone. Si es necesaria más información referente a esta materia remitimos al interesado a cualquiera de los manuales de C++ que aparecen en el apartado de bibliografía de este documento.

## 7.3 CLASES DE LA APLICACIÓN CLIENTE

En este apartado de la memoria vamos a detallar las distintas clases que componen esta aplicación, junto con la estructura que las clasifica.

### 7.3.1 LAS CLASES CClienteApp Y CClienteDlg

Estas son las clases que contienen el programa principal. Cuando se dice que una clase está definida en el programa principal quiere decir que está contenido en estas clases. La definición de la clase CClienteDlg es la siguiente:

```
// ClienteDlg.h : header file
//
#include "MiSocket.h"
#include "Imagen.h"
#if
!defined(AFX_CLIENTEDLG_H__DACEB5BA_11B9_4A26_8A93_DC07E67
2164F__INCLUDED_)
#define
AFX_CLIENTEDLG_H__DACEB5BA_11B9_4A26_8A93_DC07E672164F__I
NCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CClienteDlg dialog

class CClienteDlg : public CDialog
{
// Construction
public:
   int longitud;

   CImagen* pImagenSave;

   CClienteDlg(CWnd* pParent = NULL);      // standard constructor

   CMiSocket *sck;//Objeto de la clase CMiSocket encargado de las
   //comunicaciones con el Servidor.
```

```
    CImagen *pImagen;//Objeto de la clase que se encarga de la //representación
    de las imágenes.

    // Dialog Data
    //{{AFX_DATA(CClienteDlg)
    enum { IDD = IDD_CLIENTE_DIALOG };
    CButton        m_Camara;
    CStaticm_Picture;
    CString        m_Directorio;
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CClienteDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    //        DDX/DDV
    support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CClienteDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnConectar();
    afx_msg void OnGuardarImagen();
    afx_msg void OnVerImagen();
    afx_msg void OnImagenCamara();
    afx_msg void OnSolicitarImagen();
    afx_msg void OnSiguienteImagen();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

```
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif
//!defined(AFX_CLIENTEDLG_H__DACEB5BA_11B9_4A26_8A93_DC07E6
72164F__INCLUDED_)
```

Esta es la clase que trata directamente con el interfaz gráfico, o lo que es lo mismo, con en diálogo. Los datos miembro más destacados que presenta esta clase son:

- *longitud (entero)*: Indica la longitud en caracteres de la ruta del archivo solicitado.
- *sck (puntero a objeto de la clase CMisocket)*: La clase CMiSocket, como se verá más adelante es la encargada de las comunicaciones entre Cliente y Servidor y a través de este objeto sck podemos acceder a las funciones de comunicación con el Servidor.
- *pImagen (puntero a objeto de la clase CImagen)*: La clase CImagen implementa funciones que nos permiten trabajar con DIBs y por tanto ofrece también funciones para el tratamiento y represtación de archivos en formato BMP. El objeto pImagen es el que nos va a permitir la representación de las imágenes que recibimos en el interfaz de usuario.

Las funciones miembro más destacadas que ofrece esta clase son:

- *virtual BOOL OnInitDialog() :* Esta función se ejecuta en el momento en que se abre la aplicación. En esta función se inicializan los objetos miembro anteriormente descritos y se llama a la función Create() de la clase CMiSocket que se encarga de crear el Socket para la comunicación.
- *afx_msg void OnPaint()*: La función OnPaint es llamada por el programa cada vez que es necesario refrescar el diálogo de la aplicación. Puesto que nuestro programa debe representar la última imagen recibida de forma permanente hasta que se reciba otra, en esta función se hace la comprobación de

una bandera que indica la presencia o no de una imagen en el buffer de recepción y si la hay, entonces se encarga de representarla en cada una de las llamadas de refresco del diálogo.

- *afx_msg void OnConectar()*: Esta función se encuentra asociada al botón Conectar del interfaz de usuario. La misión de ésta es tomar la dirección IP que el usuario ha introducido en el campo correspondiente (y que indica la dirección de la red donde se encuentra funcionando el programa Servidor) y almacenarla en una variable local de tipo CString. El siguiente paso consiste en capturar de la misma forma el número de puerto especificado por el usuario para comunicar ambas aplicaciones. Finalmente se llama a la función Conectar() de la clase CMiSocket a la que se indican como parámetros los datos anteriormente capturados para que realice una conexión vía socket con las características especificadas por el usuario.

- *afx_msg void OnGuardarImagen()*: Se trata de la función que se ejecuta cuando el usuario pulsa en el diálogo el botón Guardar Imagen. Ésta se encarga de abrir un diálogo de Windows correspondiente a guardar imagen donde el interesado podrá especificar el nombre y la ruta de la imagen que desea almacenar. Podrá crear una nueva o bien sobrescribir una imagen existente. La imagen que se almacena es la última que se ha recibido desde la aplicación Servidor, ya sea desde un archivo, una imagen de una secuencia o bien una imagen recibida de la cámara. Esta imagen estará en formato BMP. Para realizar este proceso se emplean funciones de la clase CImagen que más adelante se detallarán.

- *afx_msg void OnVerImagen()*: Esta función hace la operación contraria a la anterior. Cuando se pulsa el botón correspondiente a Ver Imagen se abre el diálogo de Windows que corresponde a Abrir Archivo donde el usuario especifica la ruta y el archivo a abrir. Una vez abierto el archivo es mostrado en la zona de visualización de imágenes del interfaz de usuario. La imagen ha de estar en formato BMP. De nuevo se emplean funciones de la clase CImagen que se describen posteriormente en este capítulo.

- *afx_msg void OnImagenCamara()*: Esta función solicita al Servidor la imagen que en ese momento esté grabando la cámara. Está asociada al botón

correspondiente. La forma de operar es: en primer lugar refresca el cuadro de diálogo eliminando la imagen anterior que hubiera en la representación. A continuación utiliza la función correspondiente de la clase CMiSocket para enviar el mensaje adecuado al Servidor.

- *afx_msg void OnSolicitarImagen()*: Esta función se encarga de solicitar al Servidor la imagen especificada en el campo ruta cuando el usuario la demanda pulsando el botón del diálogo asociado a esta función. Si la imagen o el directorio no son los correctos el programa lo indicará con un mensaje de error. El proceso a seguir consiste en primer lugar en la actualización del área de representación de la imagen. Tras esto se realiza la captura de los datos introducidos por el usuario en el campo ruta, determinándose a su vez la longitud en caracteres de este campo; que será información de utilidad para el Servidor. Finalmente se envía el correspondiente mensaje especificando ruta y longitud de la ruta empleando la facilidad que ofrece la clase CMiSocket para este menester.

- *afx_msg void OnSiguienteImagen()*: Asociada al botón Siguiente Imagen, esta facilidad tiene como propósito solicitar la siguiente imagen de la secuencia que el usuario ha especificado en el campo ruta. Esto evitará al usuario tener que escribir el nombre de la siguiente imagen ya que mediante esta función, en el Servidor esta ruta se actualiza automáticamente, gracias a un algoritmo que describiremos en el siguiente capítulo.

### 7.3.2 LA CLASE CMiSocket

Esta clase es la más importante de este proyecto fin de carrera ya que en ella se sustentan todas las comunicaciones entre el Cliente y el Servidor. Es una clase derivada de la clase CSocket , ya que conserva numerosas características en común con la citada clase padre. Las posibilidades de operación que nos ofrece esta clase quedan perfectamente descritas por las funciones que la conforman y que se describen más adelante. La declaración de esta clase es la siguiente:

```
#include "Imagen.h"
#include "SubDialog.h" // Added by ClassView
```

```
#if
!defined(AFX_MISOCKET_H__F17C5312_9123_4FAD_A550_25ECF84E2BF
6__INCLUDED_)
#define
AFX_MISOCKET_H__F17C5312_9123_4FAD_A550_25ECF84E2BF6__INCL
UDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MiSocket.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CMiSocket command target
class CMiSocket : public CSocket
{
// Attributes
public:

// Operations
public:
   CMiSocket();
   virtual ~CMiSocket();
// Overrides
public:
   char* PBufferSave;//Puntero que apunta a la última imagen recibida
   int FlagImage;//Bandera que nos sirve para indicar a OnPaint si hay
//alguna imagen para representar en la visualización o no.
```

```
    SubDialog subdlg;
    char GlobalBuffer[1400000]; //Buffers que almacenan la imagen //recibida y la
procesan
    char GlobalBuffer3[1400000];
    char GlobalBuffer2[1400000];
    long int IndiceGlobal; //Para recorrer el Buffer que almacena la //imagen
recibida
    long int ByteFinal;//Valor que indica el byte donde termina el bloque
    bool EnviarMensaje(long int nummens, int acc, int tipo,int longitud);
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMiSocket)
    virtual int Send(const void* lpBuff, int nBuffLen, int nFlags = 0);
    virtual void OnReceive(int nErrorCode);
    //}}AFX_VIRTUAL
    void Conectar(LPTSTR IP,UINT Port);

    // Generated message map functions
    //{{AFX_MSG(CMiSocket)
            // NOTE - the ClassWizard will add and remove member //functions
here.
    //}}AFX_MSG

// Implementation
protected:
    long int Num_Mens_CS;
};
/////////////////////////////////////////////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations //immediately before the previous line.

#endif // !defined(AFX_MISOCKET_H__F17C5312_9123_4FAD_A550_25ECF84E2BF6__INCLUDED_)
```

Los datos miembro de esta clase son:

- *PBufferSave (Puntero a carácter):* Este dato miembro es un puntero que apunta en todo momento al buffer que contiene la última imagen recibida, con el fin de que cuando el usuario indique a través del diálogo su intención de guardar la imagen en un archivo en disco, el programa sepa la información que debe almacenar.

- *FlagImage (entero):* Esta bandera nos informa de si existe o no imagen recibida en los buffers de recepción. El objetivo es que cuando se llame a la función OnPaint() para refrescar el diálogo de la aplicación, podamos saber si hemos de volver a mostrar una imagen que el usuario estaba visualizando o no.

- *subdlg (objeto de la clase CDialog)*: Este objeto es invocado cuando se recibe un mensaje de ruta errónea por parte del servidor y se le comunica al usuario a través de un diálogo.

- *GlobalBuffer[1400000] (Cadena de caracteres):* Esta variable, junto con *GlobalBuffer2* y *GlobalBuffer3* se emplean para almacenar la información recibida y procesarla. El tamaño de estos buffers es el que nos limita el tamaño de archivo máximo con que pueden trabajar nuestras aplicaciones. La necesidad concreta para la que han sido diseñados estos programas no necesitan de mayor capacidad y consideramos innecesario reservar más memoria si no se va a utilizar. De igual forma diremos que en cualquier momento podemos aumentar la capacidad de transmisión de información con tan sólo aumentar el tamaño de estas cadenas de caracteres.

- *IndiceGlobal (entero largo)*: Cuando la información a transmitir a través del socket sobrepasa el tamaño de los 8 Kb aproximadamente, los niveles inferiores del modelo OSI que intervienen en la transmisión de la información pueden fragmentarla en paquetes más pequeños. Estos paquetes son de tamaño variable dependiendo del nivel de carga de la red en ese momento. Por este motivo, aunque la información llega ordenada debido al tipo de socket que se emplea en este proyecto y que se describe en el apartado 4.5.4 de este documento, debemos reagruparla para poder procesarla de manera correcta. Este índice sirve para llevar el recuento del número de bytes recibidos y así poder recorrer los buffers de recepción conforme se van recibiendo paquetes de información.

- *ByteFinal (entero largo)*: Indica el número de bytes que tiene el paquete de información que se ha recibido, que como hemos explicado anteriormente es de tamaño variable.

Las funciones miembro que componen esta clase son:

- *void Conectar(LPTSTR IP,UINT Port)*: La función Conectar llama a la función Connect que se encarga de abrir la comunicación entre el Cliente y el Servidor, estando este Servidor ejecutándose en el equipo con la dirección IP que se le pasa como parámetro. La comunicación se establece a través del número de puerto indicado también como parámetro.

- *bool EnviarMensaje(long int nummens, int acc, int tipo,int longitud)*: Esta función se encarga de enviar un mensaje desde el Cliente al Servidor indicándole a este último que tipo de operación solicita. Los parámetros que toma esta función es la información que se le indica al servidor:

  - *long int nummens*: Indica el número de imagen dentro de una secuencia de imágenes que se desea recibir.

  - *int acc*: Le indica al Servidor el tipo de acción que se desea realizar. Si vale 1 estamos solicitando una imagen suelta o la primera imagen de una secuencia, si vale 2 requerimos la siguiente imagen de la secuencia, si su valor es 3 pedimos una imagen de la cámara y si vale 4 estamos solicitando el número de imagen especificado en el primer parámetro dentro de una secuencia.

- *int tipo*: Es un campo que indica al servidor si el siguiente mensaje que vamos a recibir es la ruta de un archivo u otro tipo de información.

- *int longitud*: Indica al servidor la longitud del campo ruta que se va a recibir.

- *virtual int Send(const void* lpBuff, int nBuffLen, int nFlags = 0)*: Esta función se hereda de la clase padre CSocket y se encarga de enviar vía socket la información contenida en el buffer que recibe como parámetro.

- *virtual void OnReceive(int nErrorCode)*: Esta función es muy importante en el desarrollo de este proyecto fin de carrera. Esta función es llamada cada vez que se recibe información en el buffer de recepción del socket. Dicha llamada la hace Windows, es decir, nosotros no controlamos cuando se realiza cada llamada. Tan solo sabemos que lo lógico es que cuando enviamos información el punto de la red donde se ejecuta la aplicación complementaria, ya sea Cliente o Servidor (para este caso el Servidor) se realice una o varias llamadas a esta función. Es también importante reseñar que cuando enviamos una información de tamaño considerable y esta es fraccionada por el protocolo que rige el socket, la función OnReceive() es llamada cada vez que se recibe uno de estos paquetes. Es por esto por lo que hemos tenido que introducir un protocolo de nivel superior para poder indicar si la información es un paquete de datos, una imagen o bien una parte de ella. El proceso a seguir para realizar correctamente la recepción es el siguiente:

  - En primer lugar se calcula el tamaño del paquete recibido, puesto que como hemos explicado anteriormente estos son de tamaño variable en función del nivel de carga de la red. Para ello lo que se hace es buscar en el buffer de recepción una cadena de 30000 ceros.

  - A continuación y tras declarar las variables locales que se van a emplear, se analizan los primeros 9 bytes del bloque recibido. En estos bytes es donde el Servidor le indica al Cliente el tipo de información que contiene el paquete.

  - Si estos bytes que actualizan la variable condición forman la cadena de caracteres “GUARDARIM”, entonces le estamos indicando al Cliente que la transmisión de la imagen ha concluido y el Cliente muestra al usuario el

diálogo de guardar imagen de Windows para que este especifique ruta y nombre del archivo donde quiere almacenar la imagen y además se muestra esta por pantalla.

- Si la variable condición almacena la cadena "VERIMAGEN" le estamos indicando a la aplicación Cliente que la transmisión de la imagen ha concluido y que proceda a representar directamente la imagen recibida por pantalla.

- Si condición contiene la cadena "DIGITALIZ", entonces indicamos al Cliente que la imagen de procedente de la cámara ha sido enviada por completo y se le exige que la represente por pantalla. La imagen que proviene de la cámara no está en formato BMP que es el que emplea esta aplicación para representar la imagen y para almacenarla en disco. Por tanto hemos de configurar correctamente todos los campos que forman un archivo BMP empezando por los datos. Colocamos las filas en orden inverso como corresponde en el formato y tras ellos transformamos de RGB que es el formato que da la tarjeta de Imaging Technology, a BGR que es formato de los datos en un archivo BMP. Finalmente añadimos al principio del bloque la cabecera correspondiente y ya tenemos la información proveniente de la cámara en el formato deseado. Una vez hecho esto se representa la imagen.

- Si los primeros nueve bytes del paquete de información contienen la cadena "RUTAERROR", estamos indicando al programa Cliente que la ruta introducida por parte del usuario no se corresponde con ninguna existente en el ordenador donde se ejecuta el Servidor. Esta aplicación Cliente genera el correspondiente mensaje de error al usuario.

- Si no estamos en ninguno de los casos anteriores entonces se trata de un paquete de datos que forma parte de una imagen, por lo que es almacenado en los buffers de recepción actualizando convenientemente los índices que permiten recorrerlos.

### 7.3.3 LA CLASE CImagen

En líneas generales podemos decir que la clase CImagen es la que se encarga de la representación de la imágenes que llegan al Cliente. Es una clase que presenta una

amplia funcionalidad. Por tanto a la hora de realizar la descripción de sus clases solamente vamos a reseñar aquellas de mayor utilidad para nuestro propósito.

La declaración de esta clase es:

```
// Imagen.h: interface for the CImagen class.
//
//////////////////////////////////////////////////////////////////////


#if
!defined(AFX_IMAGEN_H__BDB6B44F_6516_4A62_AA05_D47885980EA4
__INCLUDED_)
#define
AFX_IMAGEN_H__BDB6B44F_6516_4A62_AA05_D47885980EA4__INCLU
DED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
class CImagen : public CObject
{
   enum Alloc {noAlloc, crtAlloc, heapAlloc};
   DECLARE_SERIAL(CImagen)
public:
   LPVOID m_lpvColorTable;
   HBITMAP m_hBitmap;
   LPBYTE m_lpImage;  // dirección de comienzo de los bits del DIB
   LPBITMAPINFOHEADER m_lpBMIH; //  búfer que contiene la
//BITMAPINFOHEADER
private:
```

```
    HGLOBAL m_hGlobal; // para ventanas externas necesitamos liberar;
                       //  podría ser asignada por esta clase o externamente
    Alloc m_nBmihAlloc;
    Alloc m_nImageAlloc;
    DWORD m_dwSizeImage; // de bits -- no BITMAPINFOHEADER o
//BITMAPFILEHEADER
    int m_nColorTableEntries;


    HANDLE m_hFile;
    HANDLE m_hMap;
    LPVOID m_lpvFile;
    HPALETTE m_hPalette;
public:
    CImagen();
    CImagen(CSize size, int nBitCount);          //                    construye
//BITMAPINFOHEADER
    virtual ~CImagen();
    int GetSizeImage() {return m_dwSizeImage;}
    int GetSizeHeader()
            {return sizeof(BITMAPINFOHEADER) + sizeof(RGBQUAD) *
m_nColorTableEntries;}
    CSize GetDimensions();
    BOOL AttachMapFile(const char* strPathname, BOOL bShare = FALSE);
    BOOL CopyToMapFile(const char* strPathname);
    BOOL AttachMemory(LPVOID lpvMem, BOOL bMustDelete = FALSE,
HGLOBAL hGlobal = NULL);
```

```
    BOOL Draw(CDC* pDC, CPoint origin, CSize size);  // hasta que
implementamos CreateDibSection
    HBITMAP CreateSection(CDC* pDC = NULL);
    UINT UsePalette(CDC* pDC, BOOL bBackground = FALSE);
    BOOL MakePalette();
    BOOL SetSystemPalette(CDC* pDC);
    BOOL Compress(CDC* pDC, BOOL bCompress = TRUE); // FALSE
significa descomprimir
    HBITMAP CreateBitmap(CDC* pDC);
    BOOL Read(CFile* pFile);
    BOOL ReadSection(CFile* pFile, CDC* pDC = NULL);
    BOOL Write(CFile* pFile);
    void Serialize(CArchive& ar);
    void Empty();

private:
    void DetachMapFile();
    void ComputePaletteSize(int nBitCount);
    void ComputeMetrics();
};
#endif                                                    //
!defined(AFX_IMAGEN_H__BDB6B44F_6516_4A62_AA05_D47885980EA4
__INCLUDED_)
```

La mejor forma de describir la Clase CImagen es a través de las funciones miembro que la componen:

- *Constructor por Omisión*: Se utiliza en la preparación para la carga de un DIB desde un archivo o para asociar un DIB a memoria. El constructor por omisión crea un objeto DIB vacío.

- *CImagen(CSize size, int nBitCount). Constructor de la Sección de DIB*: Si necesitamos una sección de DIB que es creada por la función *CreateDIBSection* entonces empleamos este constructor. Sus parámetros determinan el tamaño y el número de colores del DIB. El constructor asigna la memoria de la cabecera de información pero no de la imagen. También se puede utilizar este constructor si necesitamos asignar nuestra propia memoria de imagen. El primer parámetro *size* es un objeto CSize que contiene la altura y anchura del DIB. El parámetro *nBitCount* indica el número de bits por píxel.

- *BOOL AttachMapFile(const char* strPathname, BOOL bShare = FALSE)*: Esta función abre un archivo proyectado en memoria en modo lectura y lo asocia al objeto CDib. Vuelve inmediatamente porque no se lee realmente el archivo en memoria hasta que se utiliza. Sin embargo, cuando se accede al DIB se podría producir un retraso mientras se lee el archivo. La función *AttachMapFile* libera la memoria asignada existente y cierra cualquier archivo proyectado en memoria asociado previamente. Los parámetros son el nombre de la ruta del archivo a proyectar, una bandera que indica si el archivo se va a abrir en modo compartido y el valor devuelto es TRUE si la operación se ha realizado con éxito.

- *BOOL AttachMemory(LPVOID lpvMem, BOOL bMustDelete = FALSE, HGLOBAL hGlobal = NULL)*: Esta función asocia un objeto CDib existente a un DIB en memoria. Esta memoria podría estar en los recursos del programa o podría ser la memoria del portapapeles o de objeto de datos OLE. La memoria asignada podría haber sido del montículo CRT con el operador *new* o del montículo de Windows con GlobalAlloc. El primero de sus parámetros indica la dirección de memoria a asociar, el segundo es una bandera que es TRUE si la clase CImagen es la responsable de borrar esta memoria. Si el segundo parámetro se ha establecido a TRUE, el objeto CImagen necesita guardar el descriptor para liberarlo más adelante si la memoria se obtuvo con una llamada a la función GlobalAlloc de Win32. El valor devuelto por esta función es TRUE si la operación se ha realizado con éxito.

- *BOOL CopyToMapFile(const char* strPathname)*: Esta función crea un nuevo archivo proyectado en memoria y copia todos los datos existentes en el CDib en la memoria del archivo, liberando previamente cualquier memoria previamente asignada y cerrando cualquier archivo proyectado en memoria. Los

datos no se escriben realmente es disco hasta que se cierra el nuevo archivo, pero eso sucede cuando el objeto CImagen se reutiliza o se destruye. El parámetro que toma esta función es la ruta del archivo que se va a proyectar. Devuelve TRUE si ha tenido éxito.

- *BOOL Draw(CDC* pDC, CPoint origin, CSize size)*: Esta función muestra el objeto CImagen en la visualización o en la impresora con una llamada a la función *StretchDIBits* de Win32. El mapa de bits se escala hasta encajarse perfectamente en el área de visualización del interfaz de usuario. El primer parámetro (*pDC*) que toma esta función es un puntero al contexto de dispositivo de visualización o impresora que recibirá la imagen del DIB. El segundo parámetro (*origin*) es un objeto CPoint que contiene las coordenadas lógicas en que se visualizará el DIB. El parámetro *size* representa la anchura y altura del rectángulo de visualización en unidades lógicas. Devuelve TRUE si la operación se ha realizado con éxito.

- *CSize GetDimensions()*: Esta función devuelve la anchura y altura del DIB en píxeles. Devuelve un objeto CSize.

- *BOOL Read(CFile* pFile)*: Esta función lee un DIB de un archivo en el objeto CImagen. El archivo debe haber sido abierto con éxito. Si el archivo es un BMP, la lectura comienza desde el principio del archivo. Si es un documento, comienza desde el puntero al archivo actual. El parámetro que toma esta función (*pFile*) es un puntero al objeto CFile que es el archivo en disco que contiene el DIB. Devuelve TRUE si tiene éxito.

- *BOOL Read(CFile* pFile)*: Si se tiene un DIB de 16,24 ó 32 bits por píxel que no tiene una tabla de colores, se puede llamar a esta función para crear una paleta lógica para el objeto CImagen que coincida con la paleta devuelta por la función *CreateHalftonePalette*. Si el programa se está ejecutando en una visualización con paletas de 256 colores y no se llama a *SetSystemPalette*, no tendremos ninguna paleta y sólo aparecerán los 20 colores estándares de Windows en el DIB. Como parámetro toma un puntero al contexto de visualización. Devuelve TRUE si tiene éxito.

- *BOOL Write(CFile* pFile)*: La función *Write* escribe un DIB de un objeto CImagen en un archivo. El archivo debe haber sido abierto con éxito. Como

parámetro toma un puntero a un objeto CFile donde se escribirá el DIB. Devuelve TRUE si tiene éxito.

Sobre el rendimiento de la visualización de DIB diremos que el procesamiento optimizado de DIB es ahora una característica fundamental de Windows. Las tarjetas de video modernas tiene buffers de marco que observan el formato de imagen DIB estándar. Si tenemos instaladas una de estas tarjetas, nuestros programas pueden aprovechar el nuevo motor de DIB de Windows, que acelera el proceso de dibujar directamente a partir de DIB. Si no es así los programas no funcionarán tan rápido.

### 7.3.4 LA CLASE SubDialog

De esta clase solamente hemos de decir que deriva públicamente de la clase CDialog y es la que nos sirve para mostrar por pantalla el mensaje de error correspondiente cuando el usuario indica una ruta o un nombre de fichero erróneo.

Puesto que no hemos añadido por nuestra parte funcionalidad adicional a esta clase y por tanto la mayor parte del código que la conforma es generado automáticamente por Visual Estudio no vamos a entrar en más detalle.

## 7.4 APLICACIÓN CLIENTE CON COMPRESIÓN

Como hemos explicado anteriormente en este capítulo, el código base de esta aplicación va a ser el mismo que en la aplicación Cliente sin compresión. Sin embargo, puesto que no es exactamente el mismo procedemos en este apartado a detallar los cambios que han sido necesarios para que los programas funcionen correctamente.

### 7.4.1 LA CLASE CCompresion

La clase Compresión es la modificación más importante que se realiza a los programas. Esta clase es la que agrupa las funciones necesarias para aplicar el método de Huffman Adaptativo (véase apartado 5.5.9) a la información que se va a transmitir. No deriva de ninguna clase padre y su declaración es la siguiente:

```
#define
AFX_COMPRESION_H__A898C657_9639_4352_94D5_63BBD42E85D9__IN
CLUDED_
```

```
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class CCompresion
{
public:
   char* DescBuff;//Este es buffer donde se encuentra la imagen ya
//descomprimida
   int getuhuff(FILE *ib);
   int getcr(FILE* ib);
   void init_huff();
   void init_cr();
   unsigned int getx16(FILE* iob);
   unsigned int getw16(FILE* iob);
   void unsqueeze(FILE* inbuff, char* infile);
   void obey(char* p);
   int gethuff(FILE *ib);
   void wrt_head(FILE *ob, char* infile);
   int buildenc(int level, int root);
   void init_enc();
   void init_tree();
   char maxchar(char a, char b);
   void bld_tree(int list[], int len);
   char cmptrees(int a, int b);
   void adjust(int list[], int top,int bottom);
```

```
    void heap(int list[], int length);
    void scale(unsigned int ceil);
    void phuff();
    void pcounts();
    void oflush(FILE *iob);
    void putwe(int w, FILE* iob);
    void putce(int c,FILE*  iob);
    int getc_crc(FILE* ib);
    void init_huff(FILE* ib);
    int getcnr(FILE* iob);
    void init_ncr();
    char* squeeze(FILE * inbuff, FILE * outbuff,char* infile,char* outfile);
    CCompresion();
    virtual ~CCompresion();
};
//#endif                                                                    //
!defined(AFX_COMPRESION_H__A898C657_9639_4352_94D5_63BBD42E8
5D9__INCLUDED_)
```

Como se desprende de su código, esta es una clase totalmente portable e integrable en otras aplicaciones dado que el usuario solamente debe conocer cómo manejar las funciones squeeze(comprimir) y unsqueeze (descomprimir).

El dato miembro de esta función es:

- *DescBuff (Puntero a carácter)*: Este dato miembro es un buffer que contiene la información una vez descomprimida y al que se accede desde el programa principal para rescatar estos datos y presentarlos al usuario.

La principales funciones miembro que componen este tipo de dato y que describen perfectamente su funcionalidad son:

- *char* squeeze(FILE * inbuff, FILE * outbuff,char* infile,char* outfile)*: Esta es la función que se encarga de comprimir la información que se haya almacenada en el archivo que se le pasa como parámetro. Los parámetros que toma esta función son los punteros a FILE de los archivos de entrada (información sin comprimir) y de salida (información comprimida) y las cadenas de caracteres que indican las rutas de los archivos origen y destino. Para poder entender el código solamente hemos de entender el procedimiento que emplea el algoritmo para trabajar y que está detallado en el apartado 5.5.8 de este proyecto.

- *void CCompresion::unsqueeze(FILE * inbuff, char * infile)*: Esta función es la que realiza la descompresión de la información previamente comprimida por el procedimiento de Huffman Adaptativo. Solamente toma como parámetros el puntero a FILE que identifica el archivo a descomprimir y la cadena de caracteres que especifica su ruta. Para entender el código que se encuentra implementado en esta función de nuevo remitimos al lector al apartado 5.5.8 de la memoria en el que se detalla la forma de proceder para recuperar por completo la información en exactamente el mismo estado que antes de la citada compresión. Si es importante destacar durante la compresión y la descompresión se sigue un control de errores mediante un checksum que nos permitiría poder indicar al usuario que ha habido un error durante el proceso.

### 7.4.2 OTRAS MODIFICACIONES EN EL CÓDIGO

Además de la introducción de la clase Compresión han sido necesarias algunas modificaciones para realizar la completa adaptación de la compresión al sistema de transmisión. La principal modificación reside en la aplicación Servidor que se describe en el siguiente capítulo.

Por parte de la aplicación Cliente hemos de decir:

- Dentro de la función OnReceive (Clase CMiSocket), en el momento en que se recibe la información de la imagen comprimida por completo, se almacena en un archivo proyectado en disco desde el que se llama a la función unsqueeze que es la que se encarga de la descompresión. Un vez terminado este proceso se accede al dato miembro de la clase CCompresion *DescBuff* que almacena la información ya recuperada para continuar con el proceso habitual de representación.

- También dentro de la función OnReceive se ha eliminado el procesado que se hacía a la imagen cuando esta provenía directamente de la cámara. El motivo es que para poder realizar la compresión, la información se le ha de proporcionar a la función squeeze desde un archivo proyectado en disco. Por ello hemos de dar el formato adecuado a la imagen antes de realizar la compresión. Este es el motivo por el cual se ha trasladado este procesado a la aplicación Servidor con compresión. Así cuando la información que llega al Cliente proviene de la cámara, esta viene comprimida pero ya en el formato adecuado y solamente hemos de indicarle al programa que la represente al usuario.

## 7.5 LA APLICACIÓN CLIENTE INTEGRADA

Como hemos descrito a lo largo de este proyecto fin de carrera, aunque las herramientas desarrolladas son una potente aplicación por sí mismas, estas se van a implementar dentro de un proyecto de mayor magnitud. Además no se descarta que en un futuro estas puedan ser empleadas por otras aplicaciones que lo necesiten.

Para facilitar las labores de integración, se ha desarrollado una clase CTransmision que incluye toda la funcionalidad que ofrece la aplicación Cliente. Es decir, mediante esta clase se crea y abre un socket que nos conecta con la aplicación Servidor y mediante los parámetros que se le pasan al constructor de la clase y las funciones miembro de esta podemos cubrir todas las necesidades de la aplicación donde es integrada.

La declaración de esta clase es la siguiente:

```
#if
!defined(AFX_TRANSMISION_H__474E10BB_6DE2_4860_B496_DA806A28
7E6E__INCLUDED_)

#define
AFX_TRANSMISION_H__474E10BB_6DE2_4860_B496_DA806A287E6E__I
NCLUDED_

#if _MSC_VER >= 1000

#pragma once

#endif // _MSC_VER >= 1000
```

```
// Transmision.h : header file
//
#include "Imagen.h"
#include <afxsock.h>
/////////////////////////////////////////////////////////////////////////////
// Transmision command target
class Transmision : public CSocket
{
// Attributes
public:
// Operations
public:

    Transmision();
    Transmision(int Acc, LPTSTR Dir, CDC* pCuadroImagen);
    virtual ~Transmision();

// Overrides
public:
    void ImagenDigital();
    CImagen* pPict;
    void PintarImagen();
    void SaveIntoFile(CString filename);
    int Longitud;
    void SiguienteImagen();
    int NumSec;
```

```
    char* Directorio;
    int Accion;
    CDC* pCuadroImagen;
    CImagen* pPicture;
    char GlobalBuffer2[1400000];
    char GlobalBuffer3[1400000];
    char GlobalBuffer[1400000];
    bool EnviarMensaje(long int nummens, int acc, int tipo,int longitud);
    int IndiceGlobal;
    void Conectar();
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(Transmision)
    virtual int Send(const void* lpBuff, int nBuffLen, int nFlags=0);
    virtual void OnReceive(int nErrorCode);
    //}}AFX_VIRTUAL

    // Generated message map functions
    //{{AFX_MSG(Transmision)
            // NOTE - the ClassWizard will add and remove member //functions
here.
    //}}AFX_MSG

// Implementation
protected:
};
/////////////////////////////////////////////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}

// Microsoft Developer Studio will insert additional declarations immediately
before the previous line.

#endif //
!defined(AFX_TRANSMISION_H__474E10BB_6DE2_4860_B496_DA806A287E
6E__INCLUDED_)
```

Como se desprende del código que implementa esta clase, al crear un objeto de esta, se crea una conexión vía socket para comenzar la transmisión de las imágenes. El nombre de los datos miembro de la clase se ha mantenido con respecto a los de la aplicación Cliente original y por tanto no vamos a redundar en su explicación.

Los parámetros que toma el constructor de la clase en este caso son:

- *Acc (entero)*: Este parámetro es un número entero que especifica la acción que se desea realizar. Las opciones que se presentan son:
  - Valor 1: Solicitamos la imagen especificada en el campo directorio.
  - Valor 2: Solicitamos una secuencia de imágenes.
  - Valor 3: Solicitamos una imagen de la digitalizadora.
- *Dir (LPTSTR)*: En este campo indicamos la ruta de la imagen o la secuencia de imágenes que solicitamos desde el programa Cliente.
- *PImagenPlace (Puntero a contexto de dispositivo)*: Este parámetro le indica a la aplicación Cliente dónde queremos que nos represente la imagen recibida.

El procedimiento que se lleva a cabo en el constructor no es más que la conexión al socket y la inicialización de los parámetros correspondientes. Tras esto se evalúa la condición que se ha introducido en el campo acción para así realizar la acción correspondiente solicitada por el usuario.

Además del nuevo constructor se han introducido o modificado las siguientes funciones miembro:

- *void OnReceive(int nErrorCode)*: Esta función sigue manteniendo la misma estructura pero presenta algunas modificaciones. Entre ellas la introducción de un nuevo mensaje fin de imagen cuando la imagen proviene de la digitalizadora. Esto es así porque la aplicación de detección de humo emplea la información de

la imagen de distintas formas. Unas veces se emplea la imagen BMP recibida con la cabecera y otras sin la cabecera. De ahí que en este caso se actualicen dos buffers con la información de esta imagen recibida con y sin cabecera. Por lo demás la función permanece igual.

- *void SiguienteImagen()*: Cuando en el constructor especificamos que queremos una secuencia de imágenes, recibimos en la aplicación Cliente la primera imagen de esta secuencia. Para solicitar el resto de imágenes que componen esta secuencia hemos de llamar a la función *SiguienteImagen()*. De esta forma no se vuelve a establecer otra comunicación entre Cliente y Servidor, sino que se aprovecha que ya está abierta y se solicitan tantas imágenes como se necesiten siempre y cuando estas pertenezcan a la secuencia inicialmente especificada.

- *void SaveIntoFile(CString filename)*: Como su propio nombre indica, esta función almacena en disco en forma de archivo la imagen recibida. La ruta del archivo donde queremos que la imagen sea almacenada se le pasa como parámetro.

- *void ImagenDigital()*: Al igual que ocurría con la secuencia de imágenes, una vez creado el objeto de la clase CTransmision y con ello abierta la conexión, si vamos a solicitar varias imágenes de la cámara no es necesario abrir de nuevo esta comunicación para cada imagen. Sencillamente llamando a la función *ImagenDigital* procederemos a recibir otra imagen.

- *void PintarImagen()*: Esta función es específica para la aplicación de detección de humo. Su misión consiste únicamente en abrir la imagen almacenada en el archivo Base.bmp y la representa en la visualización.

En virtud de lo detallado anteriormente, podemos decir que el funcionamiento de esta clase es bastante transparente al usuario ya que este solamente ha de conocer los parámetros que ha de pasar al constructor y el manejo de tres funciones miembro de la clase. Esto facilita en gran medida la tarea de integración de este proyecto en otras aplicaciones que necesiten de esta funcionalidad.

## 7.6 EL INTERFAZ DE USUARIO DE LA APLICACIÓN CLIENTE

Otro de los trabajos que se han llevado a cabo para realizar este proyecto fin de carrera ha sido el diseño de los interfaces gráficos tanto de la aplicación Cliente como de la aplicación Servidor. Si bien el interfaz de la aplicación Servidor no es tan importante, ya que este no será manipulado habitualmente por los usuarios de estos programas, en cambio el interfaz de la aplicación Cliente sí es de relevancia ya que se tratará de diseñar de forma que la mayor parte de usuarios puedan utilizar estas herramientas.

Por tanto, uno de los objetivos del diseño ha sido la realización de un interfaz amigable para el usuario. Es decir, la simplicidad y la claridad de las descripciones de cada uno de los botones dejan perfectamente definida su función. Asimismo se acompaña a los botones con los correspondientes iconos aclaratorios que de nuevo facilitan la tarea al operario.

El interfaz de la aplicación Cliente es el siguiente:

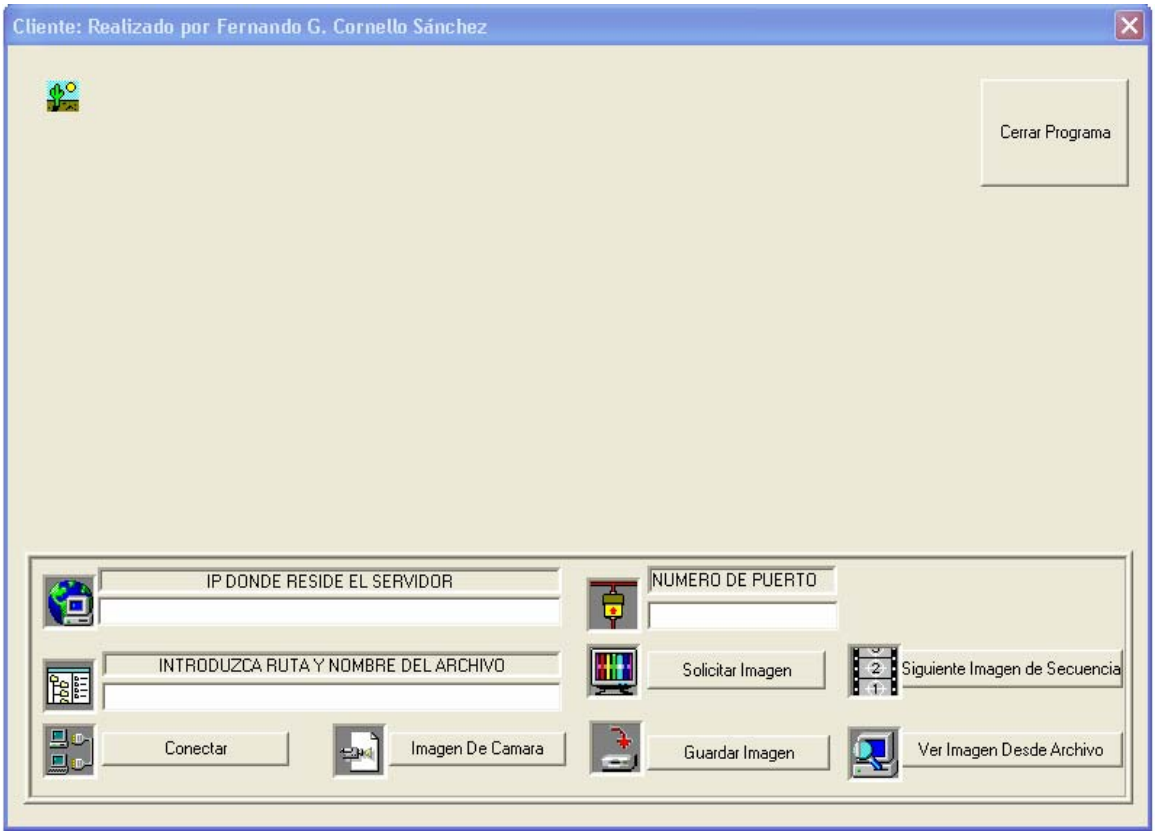



**Figura 7.1. Interfaz de la aplicación Cliente.**

Observando la figura 7.1 distinguimos dos bloques fundamentales dentro del interfaz de la aplicación: el área de representación de imágenes y la parte de control de la aplicación.

### 7.6.1 ÁREA DE REPRESENTACIÓN DE IMÁGENES

Entendemos por el área de representación de imágenes, como su propio nombre indica, a la parte del interfaz donde se muestran al usuario las imágenes recibidas.

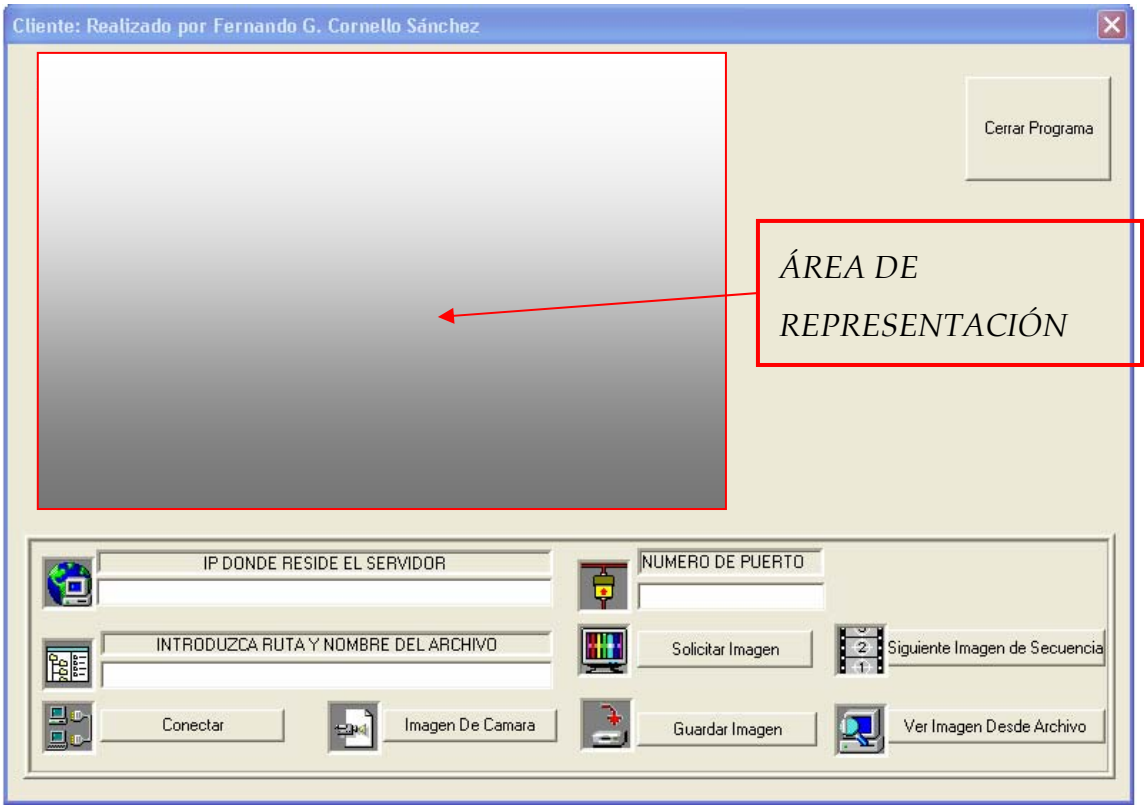



**Figura 7.2. Área de representación de imágenes del interfaz.**

Puesto que las imágenes son de muy diversos tamaños, hemos tenido en cuenta que pudiera no caber en nuestro diálogo. Por ello, previamente a la representación se realiza un reescalado de esta imagen de forma que encaje de la mejor manera posible en el área de la que disponemos. El escalado, por supuesto, mantiene las proporciones de la imagen original.

Tras las numerosas pruebas realizadas con imágenes de muy diferentes características, se ha comprobado que las dimensiones dadas a esta área de representación son adecuadas para una cómoda visualización de las mismas.

### 7.6.2 ÁREA DE CONTROL DE LA APLICACIÓN

El área de control de la aplicación es la siguiente:

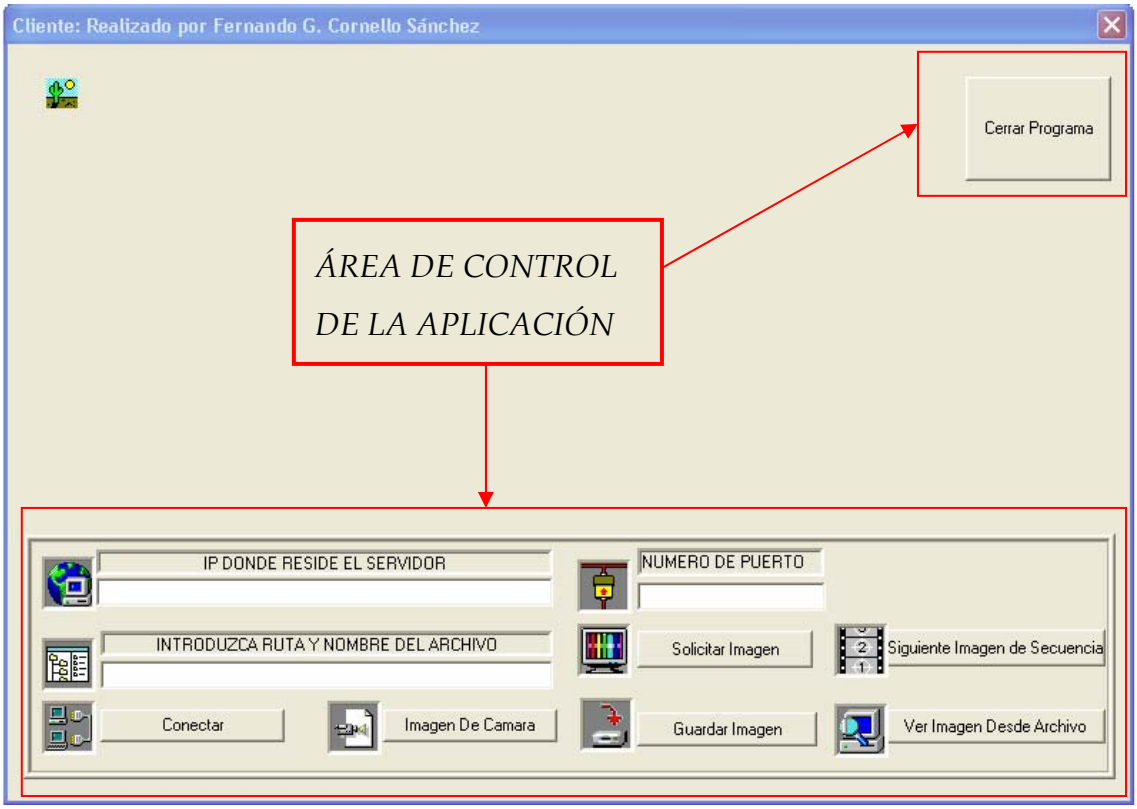
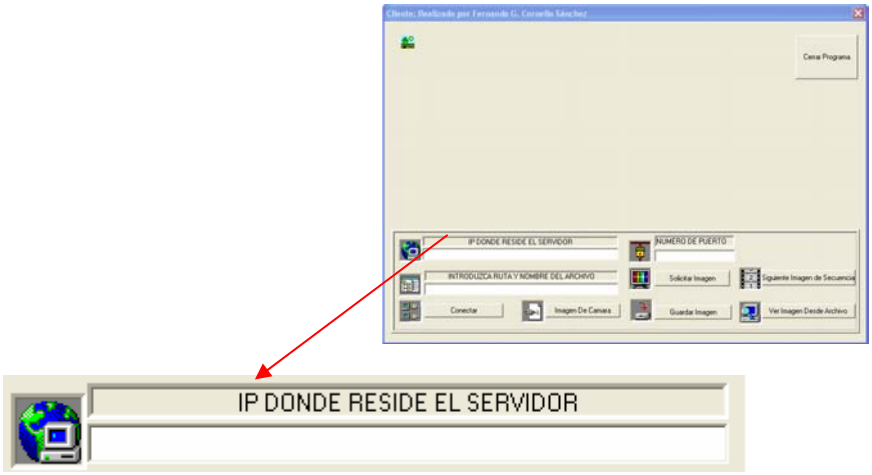



**Figura 7.3. Área de control de la aplicación.**

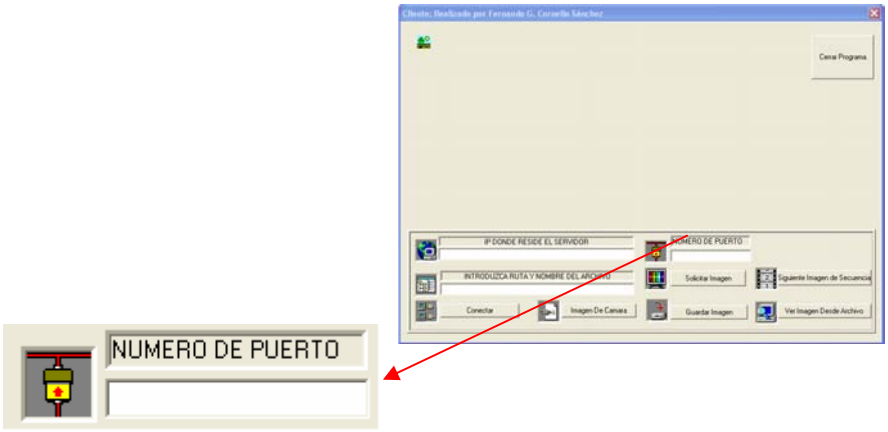
Mediante los controles situados en esta parte del interfaz de usuario podemos realizar todas las tareas para las que han sido diseñadas las aplicaciones. Existen campos donde el usuario debe introducir información y campos o botones desde los cuales este operario transmite al programa las instrucciones necesarias en cada situación. Los campos que nos encontramos son los siguientes:

- *IP DONDE RESIDE EL SERVIDOR*: Como su propio nombre indica, el operario debe introducir en este campo antes de realizar la conexión, la dirección IP dentro de la red donde se encuentra funcionando el programa Servidor. Para

ver el formato de una dirección IP remitimos al lector al apartado 4.5.4 de este documento.

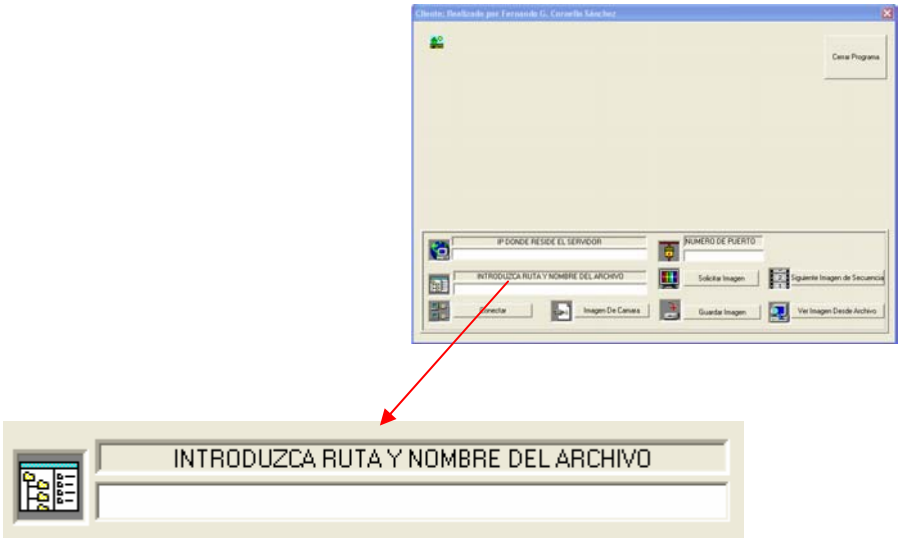



- *NÚMERO DE PUERTO*: Se trata de otro campo que debe completar el usuario antes de la conexión. En él se debe especificar el número de puerto a través del cual se quiere establecer la comunicación. Este campo resulta de especial utilidad cuando se tienen funcionando en la misma máquina varias aplicaciones que hacen uso de la red. Para más información sobre los puertos remitimos al lector al apartado 4.5.4 del presente proyecto.

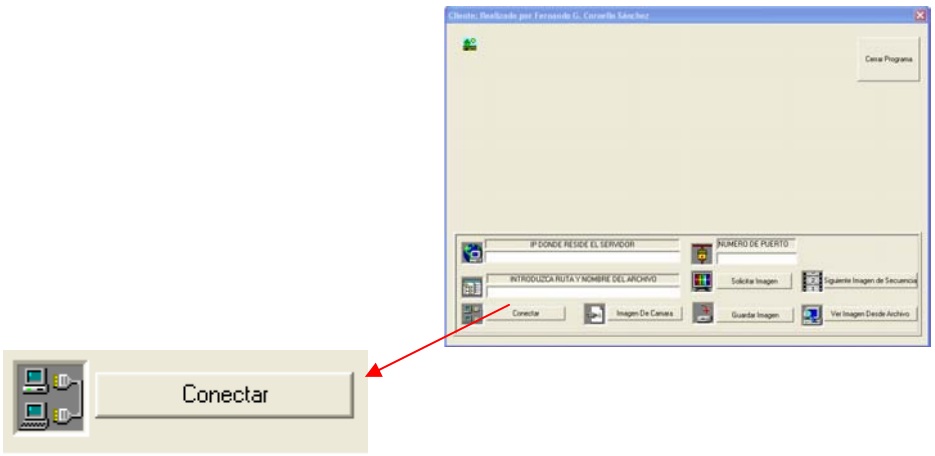



- *INTRODUZCA RUTA Y NOMBRE DEL ARCHIVO*: Tal y como se ha venido explicando a lo largo de este documento, para determinadas utilidades que

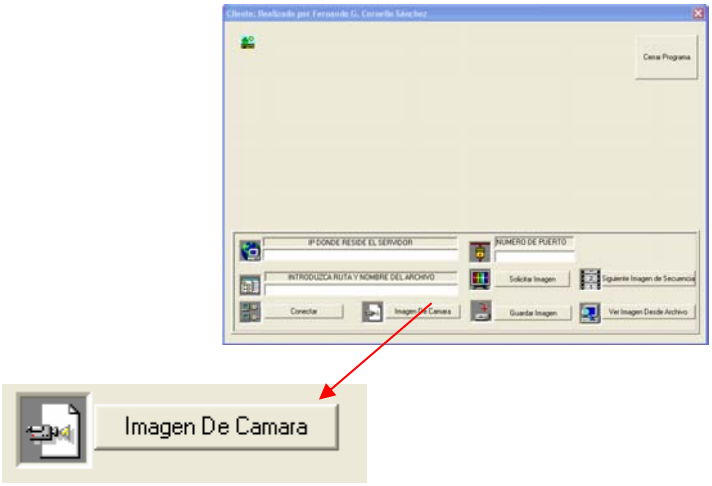
ofrecen estas aplicaciones se necesita por parte del usuario la especificación de la ruta y nombre del archivo que se pretende recibir. Es en este campo donde se permite al operario realizar esta tarea. Esta parte del interfaz, aunque pudiera parecer que limita el tamaño máximo de la ruta a introducir no es así, ya que el texto se va desplazando hacia la izquierda una vez que se ha terminado el espacio ofrecido inicialmente.



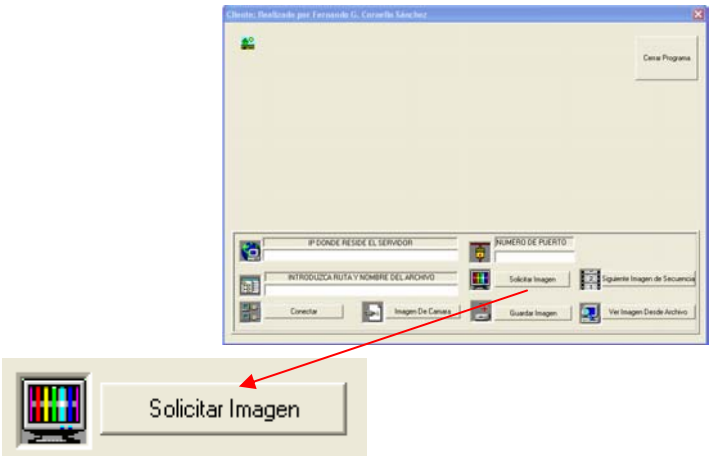

- *CONECTAR*: Este botón es el que debe activar el usuario una vez completados los campos que especifican la dirección IP y el número de puerto. La función que realiza, como su propio nombre indica es la de la conexión de las dos aplicaciones a través del socket.



- *IMAGEN DE CÁMARA*: Este botón permite al usuario solicitar la imagen que en ese preciso instante está grabando la cámara y se muestra en el área de representación.

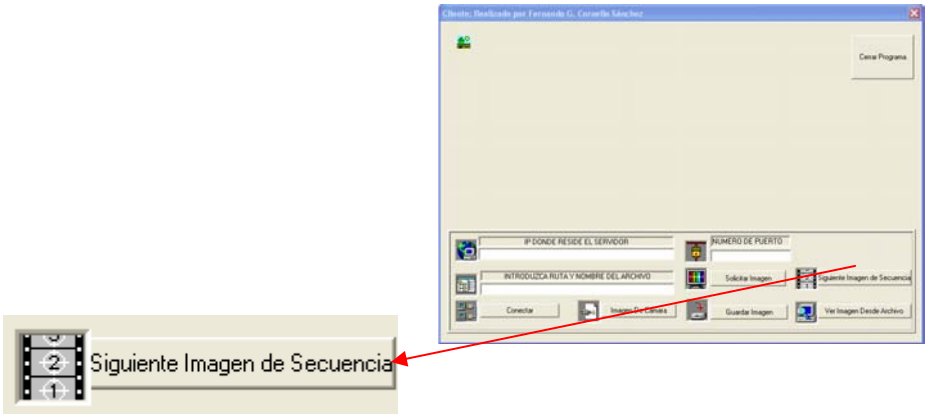



- *SOLICITAR IMAGEN*: La función que se activa cuando el usuario hace uso de esta facilidad y que le permite ver en el área de representación la imagen especificada en el campo ruta.



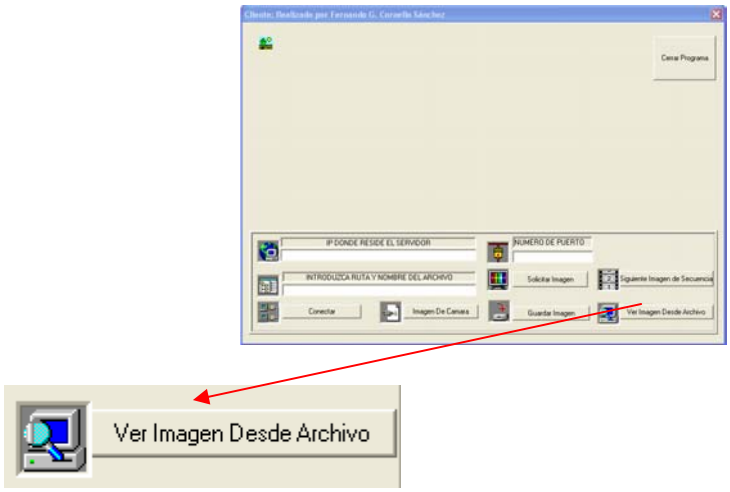

- *SIGUIENTE IMAGEN DE SECUENCIA*: Una vez solicitada la primera imagen de una secuencia, especificando la ruta correspondiente y presionando

*solicitar imagen*, podemos visualizar todas las imágenes de la secuencia una a una empleando este botón.

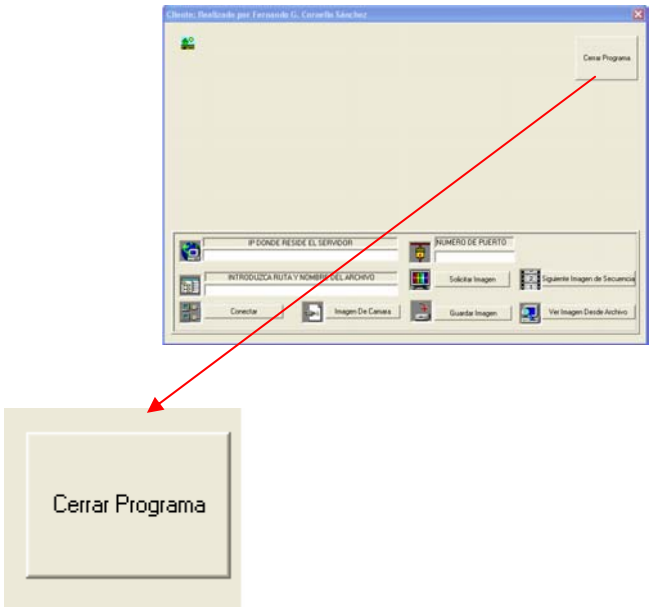


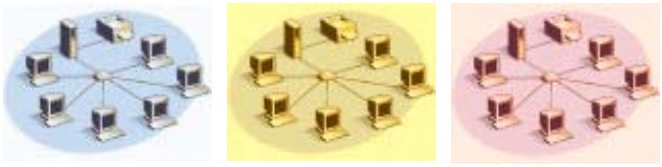

- *GUARDAR IMAGEN*: Cuando el usuario presiona este botón, se abre el diálogo de Windows correspondiente a Guardar Archivo en Disco para que el usuario pueda definir dónde quiere almacenar la última imagen recibida. Una vez hecho esto se guarda la imagen.


- *VER IMAGEN DESDE ARCHIVO*: Este componente del diálogo ofrece la posibilidad al operario de recuperar desde disco una imagen previamente guardada para visualizarla.




- *CERRAR PROGRAMA*: Este es el botón que nos permite cerrar la aplicación Cliente.





# CAPÍTULO 8

# DESARROLLO DE LA APLICACIÓN SERVIDOR

# 8 DESARROLLO DE LA APLICACIÓN SERVIDOR

## 8.1 OBJETIVOS DEL CAPÍTULO

Este capítulo es el que describe el funcionamiento de la segunda de las aplicaciones desarrolladas en este proyecto fin de carrera: la aplicación Servidor.

Puesto que este capítulo presenta la misma estructura que el capítulo anterior, no vamos a hacer de nuevo la presentación de conceptos tales como clases en C++ y similares que ya se introdujeron cuando se detalló el programa Cliente. Asimismo, puesto que como es lógico, ambos programas poseen algunas clases en común, con códigos similares, cuando se proceda más adelante a la descripción de las clases que componen esta aplicación, vamos a omitir aquellas que ya se han expuesto en el capítulo anterior.

La forma de estructurar el desarrollo durante el capítulo será, dado su mayor simplicidad, exponiendo las características de la aplicación Servidor para la tarjeta digitalizadora PC-COMP de Imaging Technology y sin compresión de información, para posteriormente detallar las modificaciones que han sido necesarias a la hora de añadir el modo de trabajo con la tarjeta Matrox y con compresión de imágenes.

Finalmente para concluir con el capítulo se realiza la descripción del interfaz de usuario de esta aplicación servidor.

## 8.2 CLASES DE LA APLICACIÓN SERVIDOR

Procedemos en este apartado de la memoria a detallar la estructura y el funcionamiento de las clases que constituyen esta aplicación Servidor. Como se ha explicado anteriormente, las clases cuya descripción se omite es porque se ha detallado en el capítulo anterior de este documento.

### 8.2.1 LAS CLASES CServidorApp Y CServidorDlg

Esta clases son las que contienen el programa principal y sobre todo en ellas se encuentran aquellas funciones que se ejecutan automáticamente al iniciar la aplicación y las que se ejecutan cuando se presiona los botones definidos en el interfaz gráfico. La definición de la clase CServidorDlg es:

```
// SevidorDlg.h : header file
```

```
//
#include "Misocket.h"
#include "Imagen.h"
#include "Secuence.h"
#include "Digitizer.h"

#if
!defined(AFX_SEVIDORDLG_H__0CD293C2_4EE0_4E23_8D6D_D493CC6
DD99B__INCLUDED_)
#define
AFX_SEVIDORDLG_H__0CD293C2_4EE0_4E23_8D6D_D493CC6DD99B__
INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
// CSevidorDlg dialog


class CSevidorDlg : public CDialog
{
// Construction
public:
   CDigitizer* digit;
   CSevidorDlg(CWnd* pParent = NULL);     // standard constructor
```

```
    CImagen *pImagen;
    CFile *miArchivo;
// Dialog Data
    //{{AFX_DATA(CSevidorDlg)
    enum { IDD = IDD_SEVIDOR_DIALOG };
            // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA


    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSevidorDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    //    //DDX/DDV
support
    //}}AFX_VIRTUAL


// Implementation
protected:
    HICON m_hIcon;


    // Generated message map functions
    //{{AFX_MSG(CSevidorDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
```

```
    afx_msg void OnButton1();
    afx_msg void OnEnviarImagen();
    afx_msg void OnEnviarMensaje();
    afx_msg void OnVerImagen();
    afx_msg void OnCapturaDigitalizadora();
    afx_msg void OnGrabacionContinua();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations //immediately before the previous line.

#endif //
!defined(AFX_SEVIDORDLG_H__0CD293C2_4EE0_4E23_8D6D_D493CC6DD99B__INCLUDED_)
```

Esta clase, al igual que su homóloga de la aplicación Cliente es la que trata directamente con el interfaz de usuario. Por ello es aquí donde se encuentran las funciones asociadas a los botones a los que puede acceder el usuario de la aplicación a través del citado interfaz.

Los datos miembros de esta clase que vamos a destacar son los siguientes:

- *sck (puntero a objeto de la clase CMiSocket)*: Es un puntero a objeto global mediante el cual accedemos a la funcionalidad de la clase CMiSocket que es la encargada de las comunicaciones con el programa Cliente.

- *digit (puntero a objeto de la clase CDigitizer)*: Este puntero a objeto es el que nos permite acceder desde la clase CServidorDlg a la clase que contiene las funciones que manejan la tarjeta digitalizadora.

- *PImagen (puntero a objeto de la clase CImagen)*: La clase CImagen se incluye de nuevo en el software Servidor. La forma de utilizar su funcionalidad es mediante este puntero a objeto de la clase.

En cuanto a las funciones miembro diremos que hemos considerado oportuno la inclusión de cierta funcionalidad accesible al usuario en el interfaz Servidor. A priori, esta posibilidad de actuación sobre la aplicación no es necesaria para el funcionamiento del conjunto, pero en algunos casos, sobre todo cuando se trata de probar la correcta instalación de los programas puede resultar de utilidad. Estas funciones posibilitan el poder visualizar las imágenes en el Servidor antes de enviarlas o incluso poder ver las imágenes que está capturando la cámara en un determinado momento, para así poder confirmar que ésta está bien calibrada o configurada para un funcionamiento adecuado. Por esto hemos de destacar las siguientes funciones:

- *virtual BOOL OnInitDialog()*: Esta función se ejecuta en el momento en que se abre la aplicación. En esta función se inicializan los objetos creados al comenzar la aplicación a la vez que se invoca a las funciones de inicialización de la tarjeta digitalizadora a través del objeto *digit*, se crea el socket y se llama a la función *Listen()* que es la que activa el modo de escucha del programa a través del socket, a la espera de recibir información por parte de un Cliente.

- *afx_msg void OnEnviarImagen()*: Como su propio nombre indica esta función abre el diálogo *explorar* al usuario que se encuentre actuando sobre la aplicación Servidor para que este pueda seleccionar la imagen que desea enviar al Cliente. Una vez seleccionada se abre de forma automática en el ordenador donde reside la aplicación Cliente el mismo diálogo, pero en este caso para poder guardar la imagen recibida. Finalmente se muestra en el área de visualización del diálogo del Cliente la imagen transmitida.

- *afx_msg void OnVerImagen()*: Esta función se encarga de abrir un diálogo de abrir archivo con el objetivo de que el operario de la aplicación Servidor pueda abrir una imagen para su visualización en el área de representación de esta

aplicación Servidor. Esta funcionalidad permite ver la imagen antes de transmitirla con el objetivo de poder comprobar, por ejemplo, que la imagen se ha trasmitido sin errores.

- *afx_msg void OnCapturaDigitalizadora()*: Al invocar esta función, se captura la imagen que en ese preciso momento está grabando la cámara conectada a la tarjeta digitalizadora y se muestra el área de representación del interfaz de usuario de la aplicación Servidor.

- *afx_msg void OnGrabacionContinua()*: Finalmente la función *OnGrabacionContinua*() permite mostrar por pantalla el proceso de grabación que está realizando la cámara en formato de video real.

### 8.2.2 LA CLASE CSubSocket

Esta clase es la encargada de recibir y analizar los mensajes que provienen del ordenador que actúa como Cliente además de enviar la información de vuelta. La definición de esta clase es la siguiente:

```
#if
!defined(AFX_SUBSOCKET_H__84D858F6_7A3C_47F2_9D8E_56E5011410
FE__INCLUDED_)

#define
AFX_SUBSOCKET_H__84D858F6_7A3C_47F2_9D8E_56E5011410FE__INC
LUDED_


//#include "Secuence.h"

#include "Digitizer.h"    // Added by ClassView

#if _MSC_VER >= 1000

#pragma once

#endif // _MSC_VER >= 1000

// SubSocket.h : header file

//

class CSecuence;
```

```
//////////////////////////////////////////////////////////////////////////
// CSubSocket command target
class CSubSocket : public CSocket
{
// Attributes
public:
// Operations
public:
    CSubSocket();
    virtual ~CSubSocket();
// Overrides
public:
    int MensaFlag;
    CDigitizer* Dig;
    CSecuence* Sec;
    int Numero_Imagen;          //Para indicar el numero que corresponde en la secuencia de imagenes
    char* Directory;        //Para almacenar la ruta de donde hemos de extraer la secuencia de im.
    int Longitud;               //Longitud de la cadena que indica la ruta.
    virtual int Send(const void* lpBuf, int nBufLen, int nFlags=0);
    virtual void OnAccept(int nErrorCode);
    int TipoMensaje;
    int Accion;
    long int NumeroSecuencia;
    virtual void OnReceive(int nErrorCode);
    virtual void OnClose(int nErrorCode);
```

```
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSubSocket)
    //}}AFX_VIRTUAL

    // Generated message map functions
    //{{AFX_MSG(CSubSocket)
        // NOTE - the ClassWizard will add and remove member //functions here.
    //}}AFX_MSG

// Implementation
protected:
};
/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations //immediately before the previous line.

#endif // !defined(AFX_SUBSOCKET_H__84D858F6_7A3C_47F2_9D8E_56E5011410FE__INCLUDED_)
```

Empezaremos, como es preceptivo, por la descripción de los datos miembro que conforman esta clase:

- *Dig (puntero a objeto de la clase CDigitizer)*: Mediante este objeto accedemos a las funciones miembro de la clase a la que pertenece la cual nos proporciona la funcionalidad de la tarjeta digitalizadora.

- *Sec (puntero a objeto de la clase CSecuence)*: La clase CSecuence, como se verá más adelante, se encarga del tratamiento de las rutas en las secuencias de imágenes. La forma de acceder a su funcionalidad desde la clase CSubSocket es mediante este dato miembro (*sec*).

- *MensaFlag (entero)*: Esta bandera nos indica si la información que se va a recibir en el próximo mensaje del Cliente es un comando o bien una ruta desde la cual extraer la imagen a transmitir.

- *Directory (puntero a carácter) y Longitud (entero)*: Indican respectivamente la ruta y la longitud de la ruta en caracteres desde donde hemos de extraer la imagen o la secuencia de imágenes que nos ha solicitado la aplicación Cliente.

Las funciones miembro que componen esta clase son similares a las del mismo nombre que se describieron en el capítulo anterior. Por tanto vamos a centrarnos en la función que soporta el funcionamiento de este programa y que es:

- *virtual void OnReceive(int nErrorCode)*: Esta función es a la que se llama cada vez que se recibe un nuevo mensaje enviado desde el Cliente. En primer lugar se realiza la comprobación de si se trata un mensaje de comando o bien un mensaje en el que se nos va a indicar la ruta desde la cual extraer la imagen a transmitir. Si se trata de la información de la ruta (mensaflag = 1) entonces se procede a actualizar la variable *Directory* que es donde se va a almacenar esta información. Se actualizan las banderas correspondientes y se establece el número de imagen de la secuencia a transmitir a cero. Finalmente se invoca a la función *PrimeraImagen* de la clase *CSecuence* que se detalla más adelante en este capítulo. Por otro lado, si el mensaje recibido no es la información de ruta sino que es un comando (mensaflag = 0), en primer lugar se inicializa el buffer de recepción a cero. Tras ello se extraen los parámetros que nos transmite el Cliente mediante la estructura de mensaje definida. A continuación se evalúa el tipo de acción que se nos especifica y se actúa en función de este valor:

  - Acción = 1: Se nos indica que la ruta de la imagen a transmitir será transmitida en el siguiente mensaje. De esta forma se prepara al Servidor para recibir esta información actualizando las banderas y liberando el buffer de recepción.

- Acción = 2: Se solicita la siguiente imagen de una secuencia cuya ruta ya ha sido especificada en un mensaje previo. En este caso no hace falta actualizar la información de ruta a partir de la información del Cliente, sino que es el propio programa Servidor el que actualiza esta ruta empleando el parámetro *Numero_Imagen* que indica el orden de imagen dentro de la secuencia que corresponde transmitir. De nuevo liberamos el buffer para una futura recepción.

- Acción = 3: Recibimos el comando de transmitir la imagen que la digitalizadora está capturando en este preciso instante.

- Acción = 4: Finalmente contemplamos el caso en que se solicita un orden de imagen especificado por el usuario dentro de una secuencia.

### 8.2.3 LA CLASE CMiSocket

Esta clase tiene como función supervisar la conexión entre el Servidor y el Cliente, encontrándose en un nivel superior a la clase CSubSocket.

Tal y como se ha venido explicando a lo largo de este proyecto fin de carrera, cuando se transmiten grandes cantidades de información a través de un socket, esta es fraccionada en paquetes de tamaño variable que se reciben en el otro extremo de la conexión de forma ordenada. Por ello es necesario indicar al Cliente (que es en este caso el otro extremo de la conexión) cuándo se ha terminado de enviar la imagen y se puede empezar a procesar la información recibida. Es en esta clase donde se incluyen las funciones que se encargan de transmitir los correspondientes mensajes de fin de transmisión de imagen que permiten la completa sincronización de las aplicaciones. Esta funciones incluyen además un tiempo de espera que necesita el socket para transmitir completamente la imagen. En algunos casos, dentro de estas funciones, este tiempo es variable en función de la cantidad de datos que se necesiten enviar.

La declaración de la clase es la siguiente:

```
#if
!defined(AFX_MISOCKET_H__7669115C_9F95_4043_990B_9BFDE585320E
__INCLUDED_)
```

```
#define
AFX_MISOCKET_H__7669115C_9F95_4043_990B_9BFDE585320E__INCL
UDED_
#include "mensajes.h"
#include "SubSocket.h"
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// MiSocket.h : header file
//
/////////////////////////////////////////////////////////////////////////////
// CMiSocket command target
class CMiSocket : public CSocket
{
// Attributes
public:
// Operations
public:
    CMiSocket();
    virtual ~CMiSocket();
// Overrides
public:
    int FileSize;
    void RutaError();
    void ImagenDigitalizadora();
    void VerSinGuardar();
    void GuardarVer();
```

```
    CSubSocket* Ssck;
    int EnviarImagen(LPCTSTR ruta);
    bool EnviarMensaje(long int nummens, int tipo,int tamano);
    CSubSocket* cliente;
    virtual int Send(const void* lpBuf, int nBufLen, int nFlags);
    virtual void OnClose(int nErrorCode);
    virtual void OnAccept(int nErrorCode);
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMiSocket)
    //}}AFX_VIRTUAL

    // Generated message map functions
    //{{AFX_MSG(CMiSocket)
            // NOTE - the ClassWizard will add and remove member //functions
    here.
    //}}AFX_MSG

    // Implementation
    protected:
};
/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations //immediately
before the previous line.
```

```
#endif                                                    //
!defined(AFX_MISOCKET_H__7669115C_9F95_4043_990B_9BFDE585320E
__INCLUDED_)
```

Esta clase no posee datos miembros destacables. En cuanto a las funciones miembro destacaremos:

- *void RutaError()*: Esta función se encarga de enviar un mensaje al Cliente indicándole a este que la ruta especificada por el usuario es incorrecta, ya que no se encuentra una de similares características en el ordenador donde reside esta aplicación. La forma de comunicar esta información es mediante la transmisión de una cadena de caracteres de 9 bytes que el Cliente reconoce como un mensaje de error, abriendo un diálogo de alerta, que le comunica la situación al operario.

- *void ImagenDigitalizadora()*: Mediante esta función se indica al programa Cliente que se ha finalizado la transmisión de la imagen de la digitalizadora solicitada y se puede proceder a su representación y procesado. Al igual que en la función anterior, la forma de indicar a la aplicación Cliente que se ha enviado por completo una imagen procedente de la cámara es mediante un mensaje de nueve bytes conocido por él.

- *void VerSinGuardar()*: Cuando al final de la transmisión de una imagen el Servidor envía al Cliente el mensaje de 9 bytes correspondiente a esta función, estamos indicándole que proceda a representar la imagen por pantalla pero sin almacenarla en disco de forma automática. Se podrá almacenar en disco posteriormente si el usuario de la aplicación Cliente lo estima oportuno.

- *void GuardarVer()*: Esta función, al contrario que la anterior, envía un mensaje de terminación de imagen otra vez de 9 bytes indicando a la aplicación Cliente que proceda a abrir de forma automática el menú correspondiente a guardar imagen para que, una vez el usuario haya especificado una ruta, se proceda al almacenaje en disco de la imagen recibida. Finalmente, una vez guardada la información, se representa la imagen en el interfaz de usuario.

- *int EnviarImagen(LPCTSTR ruta)*: La función *EnviarImagen* es, como su propio nombre indica, la encargada de transmitir la imagen desde el Servidor hasta el Cliente. El parámetro que se le especifica es la ruta de la imagen que ha de enviar. Una vez conocida la ruta se procede a abrir el archivo y se evalúa la

condición de si la apertura ha sido realizada o no con éxito. Si ha sido exitosa la apertura se procede a obtener el tamaño del archivo a transmitir para así poder reservar solamente la memoria necesaria para almacenar la imagen en concreto que deseamos trasmitir. De esta forma se optimiza el uso de memoria por parte de la aplicación. A continuación leemos la información contenida en el archivo, transmitimos completamente la imagen y finalmente cerramos el archivo previamente abierto. En el caso en que la apertura del archivo no se haya realizado correctamente, se lo comunicamos al Cliente con la llamada a *RutaError*. El valor devuelto por esta función es 1 si la operación se ha realizado con éxito ó 0 en caso contrario.

El resto de funciones son similares a las explicadas en otros apartados no procede repetir o redundar en las aclaraciones.

### 8.2.4 LA CLASE CSecuence

Para implementar la función que permite a estas aplicaciones trabajar con secuencias de imágenes almacenadas en disco se ha creado la clase CSecuence. En este objeto se han agrupado como se verá más adelante todas la funciones de generación automática de rutas, búsqueda de imágenes dentro de una secuencia etc. Hemos de recordar en este punto que para que el programa pueda funcionar de forma adecuada, las secuencias de imágenes deben estar almacenadas en disco en la forma estándar. Esta forma estándar es con la que trabajan la mayoría de las tarjetas digitalizadoras y la mayoría de los programas. Consiste en guardar en disco la secuencia nombrando cada archivo con un nombre fijo e invariante para toda la secuencia, seguido por un número que indicará el orden de esa determinada imagen dentro de la secuencia completa. Por ejemplo imagen0.bmp, imagen1.bmp, imagen2.bmp y así sucesivamente.

La declaración de la clase es la siguiente:

```
// Secuence.h: interface for the CSecuence class.
//
//////////////////////////////////////////////////////////////////////
#if !defined (_____secuence_h______)
#define _____secuence_h______
```

```
#include "MiSocket.h"


#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class CSecuence
{
public:
    char* ConBarras;
    CMiSocket* Osck;
    int LongName;
    char NombreFichero[20];
    int UltimaBarra;


    CSecuence();
    virtual ~CSecuence();


    CMiSocket* SecSck;
    void PrimeraImagen(char*,int,int);
    void InicializaParametros();
    int Numero_Secuencia;
    char* GlobalDir; //Almacena la ruta
    void SiguienteImagen(int, char*,int);
};
#endif // !defined(_____secuence_h_____)
```

Los datos miembro que componen la clase son los siguientes:

- *ConBarras (puntero a carácter)*: Todo programador en C debe conocer que el carácter "\" es un carácter de escape que tiene significado por sí mismo en cualquier sentencia de código. Pero a su vez, cuando se pretende especificar una ruta de un archivo en disco es necesaria la utilización de este mismo símbolo. Por ello, para que el compilador lo interprete correctamente hemos de duplicarlo. Pero hemos considerado que la tarea de duplicarlo no corresponde al usuario de la aplicación Cliente. Por consiguiente se ha implementado una función que busca los símbolos "\" dentro de la ruta y los duplica de forma totalmente transparente al usuario. Este dato miembro almacena la ruta introducida por el usuario tras duplicar los caracteres de escape.

- *GlobalDir (cadena de caracteres)*: Este dato miembro almacena la ruta sobre la que se va a trabajar hasta que se especifique una ruta nueva por parte del Cliente. Esta ruta se actualiza en la función *PrimeraImagen*.

En cuanto a las funciones miembro hemos de destacar las siguientes:

- *void PrimeraImagen (char*,int,int)*: Esta función es llamada cuando se quiere enviar una imagen no perteneciente a una secuencia de imágenes o bien la primera imagen de una secuencia. Toma como parámetros la ruta desde donde vamos a extraer la información, la longitud en caracteres de esa ruta y el valor de una bandera de secuencia. El primer paso dentro de esta función es actualizar la información de la ruta global desde la que se va a trabajar. Tras esto se procede a enviar la imagen.

- *void SiguienteImagen(int, char*,int)*: Una vez que se ha transmitido la primera imagen de una secuencia y se solicita por parte del Cliente que se continúen enviando secuencialmente todas la que componen la citada secuencia, entonces invocamos a esta función. La forma que tiene de proceder es, en primer lugar corregir, como se ha indicado anteriormente, la cadena de caracteres que compone la ruta, duplicando los caracteres de escape. Una vez hecho esto, analizamos el contenido de la cadena para extraer la información correspondiente al nombre del archivo. Para ello buscamos la cadena de caracteres ".bmp" y recorriendo la cadena en sentido inverso hasta el carácter "\" logramos rescatar

este dato. Una vez conocido el nombre del archivo y puesto que el número de orden que corresponde en la secuencia se le pasa como parámetro a esta función, generamos la ruta correspondiente a la siguiente imagen a transmitir. El tercer parámetro que toma esta función es la longitud en caracteres de la ruta que estamos procesando. Finalmente procedemos a trasmitir la imagen.

### 8.2.5 LA CLASE CDigitizer

La clase CDigitizer agrupa las funciones que trabajan directamente con el hardware de la tarjeta digitalizadora con la que estemos trabajando. Puesto que, tal y como hemos descrito a lo largo de este proyecto, este trabajo ha sido desarrollado para trabajar con dos tarjetas distintas, ha sido de vital importancia el diseñar esta clase de la forma lo más portable posible. En este concepto radica además que no se haya incluido en la clase funcionalidad adicional de la tarjeta que nuestras aplicaciones no vayan a emplear. Así, en le diseño nos hemos ceñido rigurosamente a las funciones necesarias para que nuestros programas funcionen correctamente.

La declaración de esta clase es:

```
// Digitizer.h: interface for the CDigitizer class.
//
//////////////////////////////////////////////////////////////////////
#if
!defined(AFX_DIGITIZER_H__7E6F0C12_FA85_408A_BF91_97D2E118EFD
7__INCLUDED_)
#define
AFX_DIGITIZER_H__7E6F0C12_FA85_408A_BF91_97D2E118EFD7__INCL
UDED_
#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000


class CDigitizer
```

```
{
public:
   BYTE* dest;
   void GrabacionContinua();
   void CapturaImagen();
   void InicializarTarjeta();
   CDigitizer();
   virtual ~CDigitizer();
};
#endif                                                    //
!defined(AFX_DIGITIZER_H__7E6F0C12_FA85_408A_BF91_97D2E118EFD
7__INCLUDED_)
```

Los datos miembro que conforman esta clase son:

- *Dest (puntero a BYTE)*: Se trata de un buffer donde almacenamos la imagen capturada por la tarjeta digitalizadora.

En cuanto a las funciones miembro destacaremos:

- *void InicializarTarjeta()*: En esta función es donde se incluyen todas las sentencias de inicialización de la tarjeta. Remitimos al lector al capítulo 6 de este documento, donde se detalla el funcionamiento de las dos tarjetas para las que se han implementado estas aplicaciones, incluyendo el procedimiento de inicialización.

- *void GrabacionContinua()*: Mediante esta función mostramos en el ordenador donde reside el programa Servidor, una ventana que permite visualizar de forma continua la grabación que está realizando la cámara.

- *void CapturaImagen()*: Como su nombre indica, esta función pone a nuestra disposición la funcionalidad de capturar la imagen que en ese preciso instante está grabando la cámara y transmitirla al Cliente siguiendo el procedimiento habitual. El procedimiento es: en primer lugar se reservan e inicializan los buffers de memoria donde se va a almacenar la imagen. A continuación se

captura la imagen y se transforma a RGB. Finalmente se transmite la imagen. Es de destacar que durante todo el proceso de captura, transmisión, recepción y representación, la imagen no se almacena en disco en ningún momento, lo que acelera el proceso, optimizando el tiempo de transmisión.

### 8.2.6 LA CLASE CImagen

La clase CImagen incluida en la aplicación Servidor coincide con la de la aplicación Cliente que se describió en el capítulo anterior. Por este motivo no vamos a redundar en las aclaraciones.

## 8.3 LA APLICACIÓN SERVIDOR CON COMPRESIÓN

Al igual que la aplicación Cliente con compresión, la base del código del programa Servidor con compresión es la misma que en el caso sin compresión. El objetivo de este apartado será aclarar aquellas modificaciones que ha sido necesario introducir para que la funcionalidad de compresión de imágenes funcione de forma adecuada.

### 8.3.1 LA CLASE CCompresión

La clase CCompresion es la que agrupa todas las funciones necesarias para aplicar a los datos el algoritmo de compresión de Huffman Adaptativo (véase apartado 5.5.8). El código de esta clase coincide exactamente con el de la clase homónima de la aplicación Cliente descrita en el capítulo anterior. Por este motivo se va a profundizar en las explicaciones.

### 8.3.2 OTRAS MODIFICACIONES EN EL CÓDIGO

Además de la inclusión de la clase CCompresion han sido necesarias ciertas modificaciones en localizaciones muy concretas dentro del código de la aplicación Servidor. Estas se detallan a continuación:

- Dentro de la función *EnviarImagen* (Clase CMySocket), es necesario, una vez abiertos los archivos origen y destino proyectados en disco, llamar a la función *squeeze* (Clase CCompresión) que realiza la compresión de la información antes de ser enviada al Cliente. Una vez hecho esto y tras la pertinente reserva de memoria, procedemos a enviar la imagen que se haya almacenada en el buffer global *DeBuff*.

- La otra modificación importante del código se encuentra en la clase CDigitizer, donde, como explicamos en el capítulo anterior, es necesario antes de enviar la imagen de la cámara, darle formato de archivo BMP para que así se pueda realizar la compresión. En la función *CapturaImagen* se han incluido las líneas de código que se encontraban anteriormente en la aplicación Cliente cuando no se realizaba la compresión de la información. La funcionalidad de esta parte del código se conserva, por tanto, intacta.

## 8.4 LA APLICACIÓN SERVIDOR PARA MATROX

Para la aplicación Cliente es prácticamente indiferente, en cuanto a código, el uso de una u otra tarjeta digitalizadora. En cambio el código del programa Servidor sí que sufre ciertas modificaciones.

Puesto que desde el momento en que se empezó a desarrollar este trabajo se conocía la necesidad de trabajar con dos tarjetas digitalizadoras de características distintas, se ha desarrollado la aplicación Servidor de forma que su funcionamiento sea lo más independiente posible en lo referente a este aspecto. Por ello se ha agrupado la funcionalidad de la digitalizadora en una sola clase (CDigitizer) de forma que esta deberá ser la única parte del código que habremos de modificar para el uso de una u otra tarjeta.

La declaración de la clase CDigitizer para el caso de la tarjeta Matrox es prácticamente igual que para el caso de la tarjeta de Imaging Technology. La única diferencia radica en la introducción de una nueva función *BufferMatrox* que permite al usuario de la aplicación acceder a los buffers de imagen de la tarjeta antes de que se les dé el formato de archivo BMP.

Puesto que en el resto del código de la clase se conserva la misma estructura, cambiando únicamente el nombre de las funciones a las que se invoca para realizar una u otra función, remitimos al lector al capítulo 6 en el que se detallan las distintas funciones que permiten trabajar con ambas tarjetas (funciones de inicialización, captura y proceso).

## 8.5 EL INTERFAZ DE USUARIO DE LA APLICACIÓN SERVIDOR

Otro de los trabajos que se ha llevado a cabo durante la realización de este proyecto fin de carrera ha sido la implementación del interfaz de usuario de la aplicación Servidor. Tal y como se explicó en el capítulo anterior, el diseño de este no es tan importante como el del programa Cliente, ya que el usuario final no habrá de trabajar con la aplicación Servidor de forma continuada. En cambio hemos considerado oportuno realizar un sencillo entorno que facilitará la correcta instalación y prueba de las aplicaciones cuando se pretendan integrar de nuevo en otro sistema.

De esta forma se puede adelantar que las funciones que se van a implementar en el interfaz de la aplicación Servidor estarán orientadas a la visualización de las imágenes que se hayan presentes en el equipo que alberga la aplicación además de las que está grabando la cámara en un instante determinado. Por otra parte también se hacen accesibles para el usuario funciones que permiten iniciar la transmisión de una imagen desde el propio Servidor, con el objetivo de comprobar la correcta configuración y funcionamiento de los programas.

El interfaz de la aplicación Servidor es el siguiente:

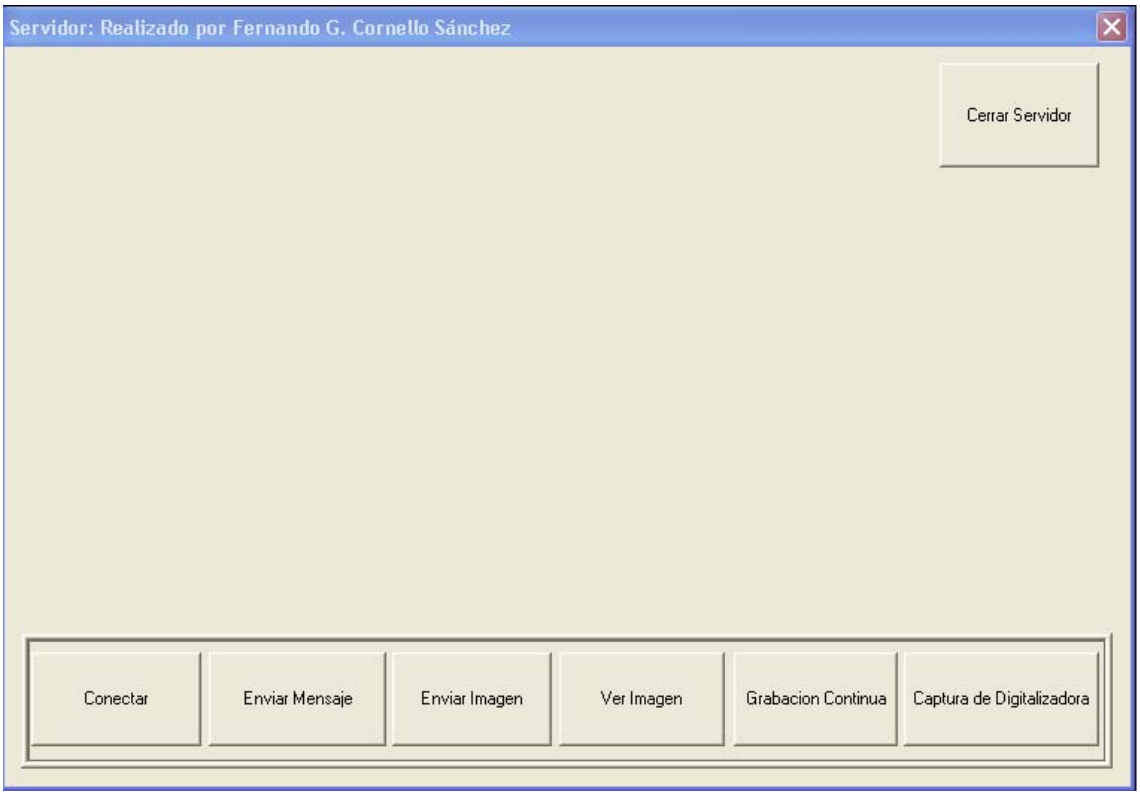



**Figura 8.1. Interfaz de usuario de la aplicación Servidor.**

De igual forma que en la aplicación Cliente encontramos dos bloques fundamentales: el área de representación de imágenes y el área de control de la aplicación.

### 8.5.1 ÁREA DE REPRESENTACIÓN DE IMÁGENES

Como su propio nombre indica, es la parte del diálogo de la aplicación sobre la que se van a representar las imágenes requeridas por el usuario.

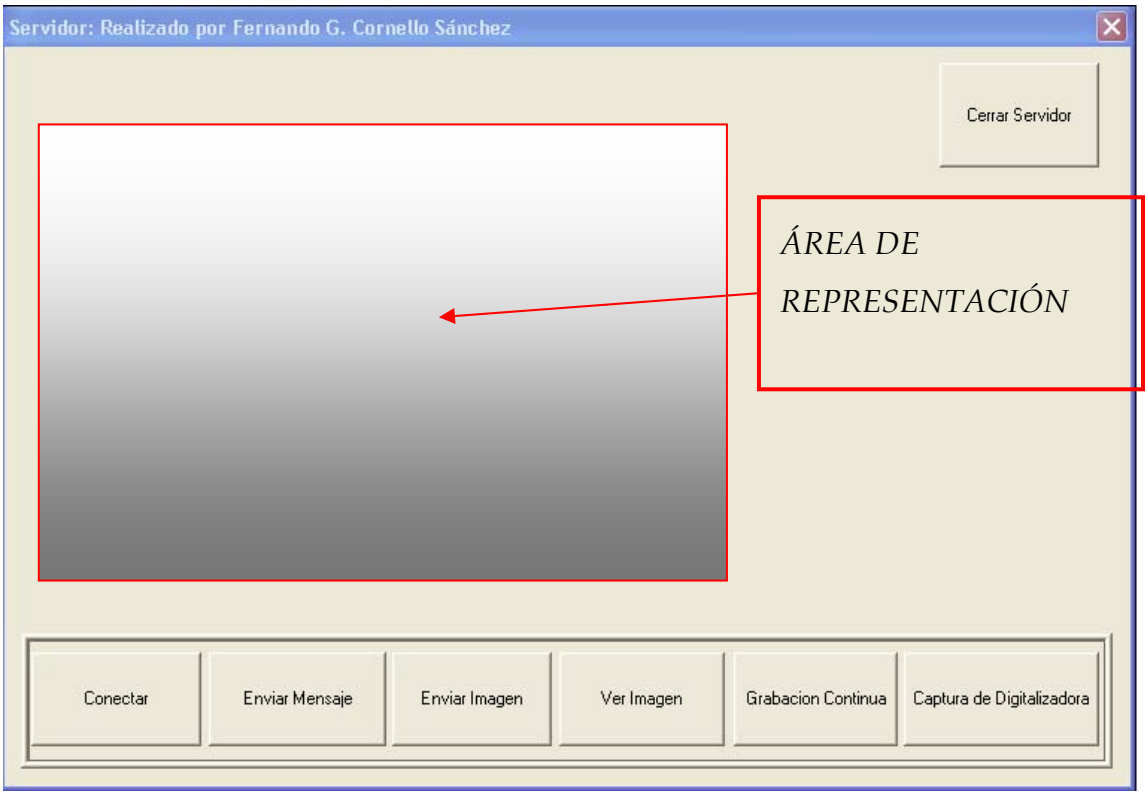



**Figura 8.2. Área de representación de imágenes del interfaz.**

Tal y como hemos explicado anteriormente, aquí se podrán mostrar tanto imágenes que se hayan almacenadas en disco en el ordenador donde reside el Servidor como imágenes que captura la cámara en el instante que indique el usuario.

### 8.5.2 ÁREA DE CONTROL DE LA APLICACIÓN

El área de control de la aplicación es aquella que contiene los botones que hacen accesible al usuario parte de la funcionalidad del programa.

Esta es la siguiente:

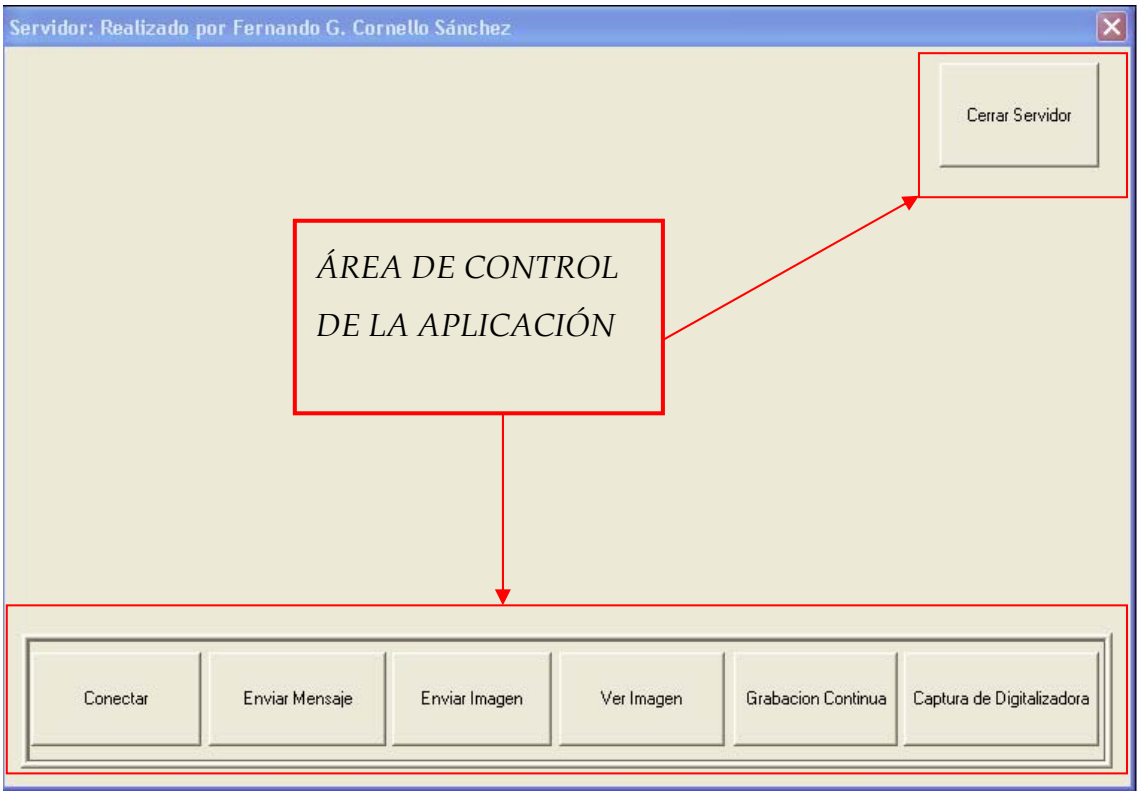
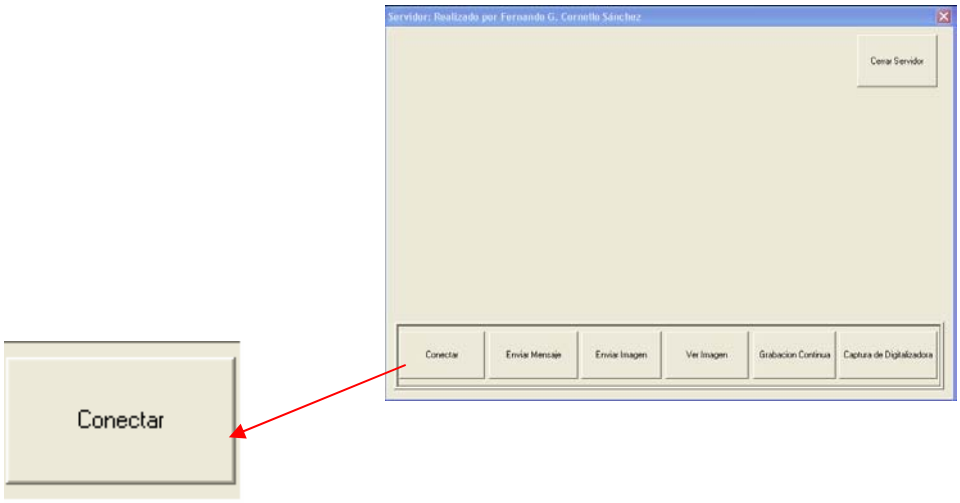


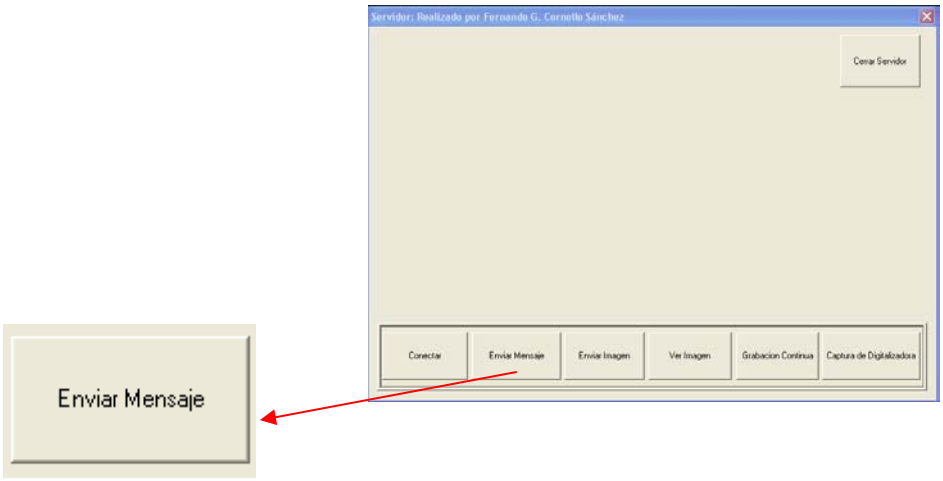

**Figura 8.3. Área de control de la aplicación.**

Como se puede comprobar en la figura 8.3, en este interfaz no existe ningún campo de texto a completar por el usuario. Solamente encontramos botones de función que son los que siguen:

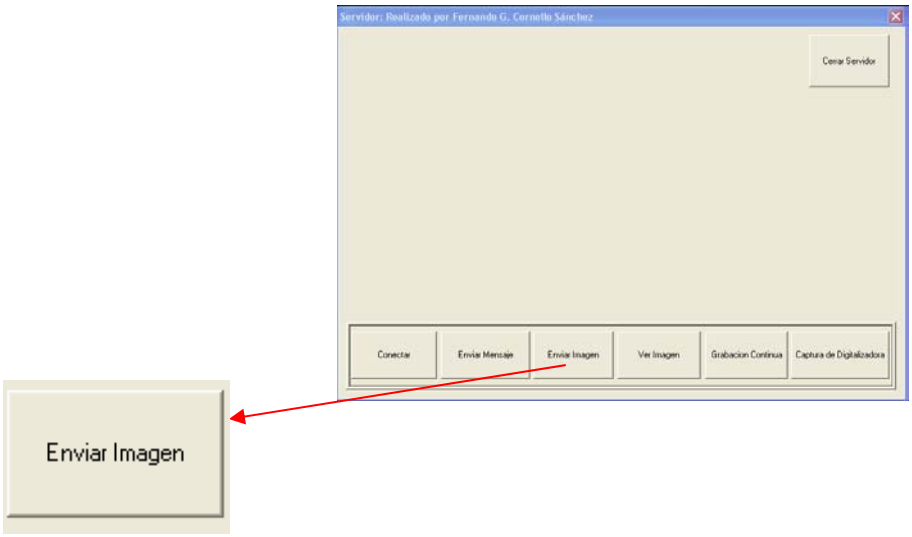
- *CONECTAR*: A priori este botón no tiene asociada ninguna funcionalidad. El objeto por el cual se ha introducido en el diseño es para el caso en que el usuario de las aplicaciones no quiera que el Servidor se conecte y se ponga a esperar peticiones en el momento de iniciar la aplicación. Es decir, el código correspondiente a la conexión y espera de peticiones se ejecuta de forma automática al arrancar la aplicación Servidor. Para modificar esto, el usuario solamente habrá de trasladar este código a la función asociada al botón Conectar para que la conexión del Servidor se realice bajo petición.



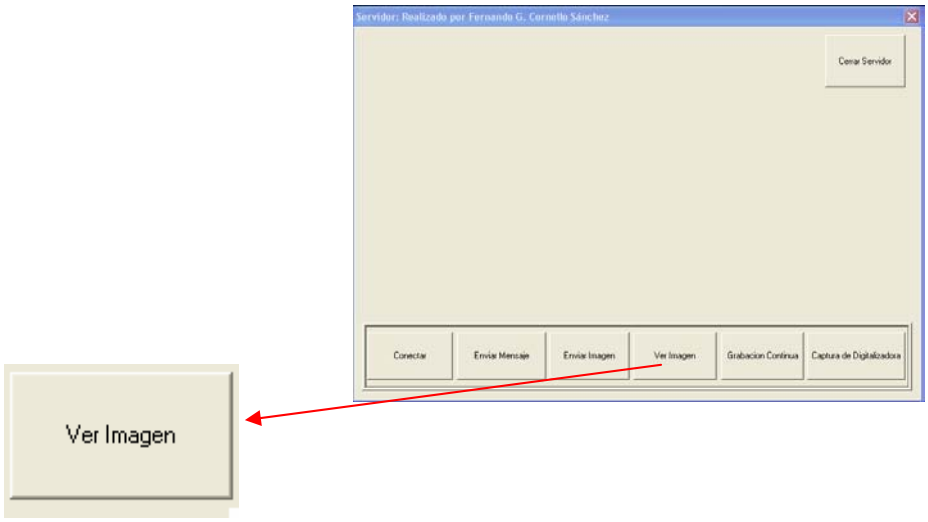

- *ENVIAR MENSAJE*: Esta función envía un mensaje de prueba a la aplicación Cliente con los campos los bytes "7","8" y "0". El objetivo de esta función no es más que la comprobación del correcto funcionamiento de la conexión y la transmisión por la red entre las aplicaciones.



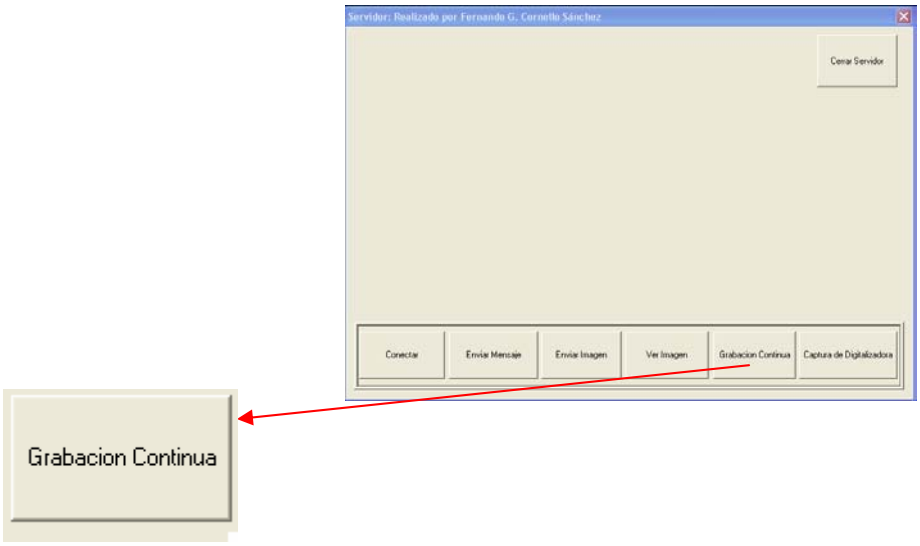

- *ENVIAR IMAGEN*: Esta función se encarga de abrir al usuario del Servidor el diálogo de abrir archivo de Windows para que éste seleccione la ruta y el nombre del archivo de imagen proyectado en disco que desea enviar al Cliente. En la aplicación Cliente se abre de forma automática el diálogo de Windows correspondiente a guardar imagen para que se especifique el nombre y el lugar donde se pretende almacenar la imagen recibida. Finalmente se muestra en el área de representación del interfaz Cliente la imagen que ha sido recibida y almacenada.



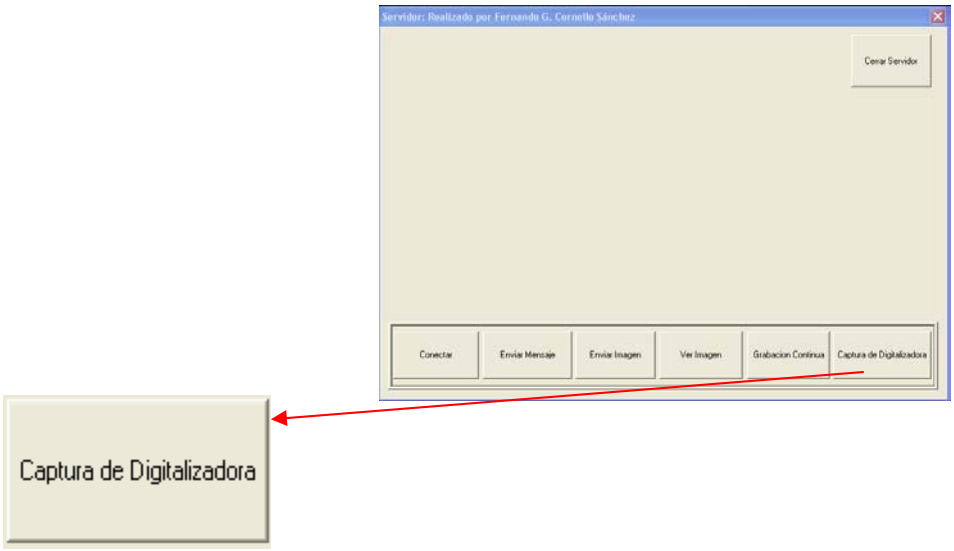

- *VER IMAGEN*: Este botón hace que se muestre por pantalla al usuario el diálogo de Windows correspondiente a abrir imagen para que este seleccione el nombre y ruta del archivo que desea visualizar en el área de representación de la aplicación Servidor. Una vez hecha la selección, la imagen es mostrada.



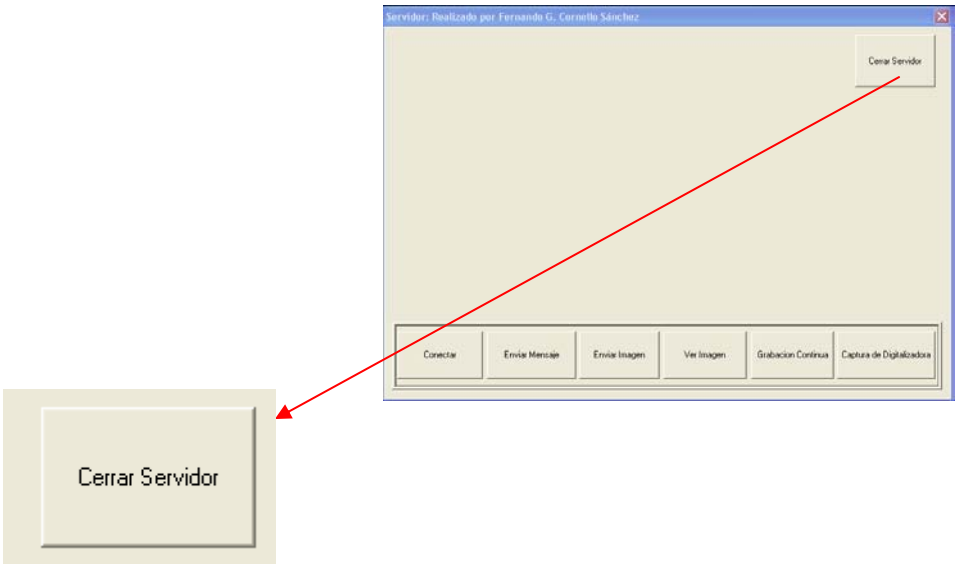

- *GRABACIÓN CONTINUA*: Mediante este botón activamos la función de la digitalizadora correspondiente que nos permite visualizar de forma continua la grabación de la cámara. Así podemos calibrar la cámara, elegir el objetivo de la grabación y otras muchas tareas previas a la transmisión de la imagen capturada.

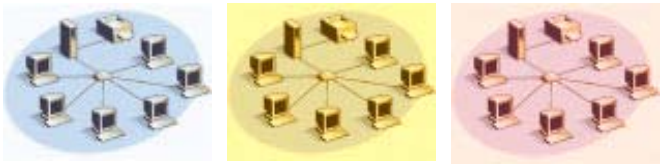



- *CAPTURA DE LA DIGITALIZADORA*: Como su propio nombre indica, este botón activa la captura de la imagen que en ese preciso instante está grabando la cámara. En el área de representación del interfaz Cliente es mostrada la imagen de forma automática.




- *CERRAR SERVIDOR*: Función que nos permite cerrar la aplicación Servidor.


Servidor: Realizado por Fernando G. Cornetto Sánchez
Cerrar Servidor
Conectar
Enviar Mensaje
Enviar Imagen
Ver Imagen
Grabacion Continua
Captura de Digitalizadora
Cerrar Servidor



# CAPÍTULO 9

# RESULTADOS Y CONCLUSIONES

# 9 RESULTADOS Y CONCLUSIONES

## 9.1 INTRODUCCIÓN

Una vez que se han explicado todos los pormenores del trabajo que se ha desarrollado, procedemos a poner de manifiesto su correcto funcionamiento a través de una serie de ensayos realizados en el entorno del laboratorio. Estos ensayos han pretendido cubrir todas las posibilidades de actuación de los diferentes programas, con el fin de poder obtener conclusiones lo más fidedignas posibles.

Posteriormente se realiza en este capítulo un análisis de los resultados obtenidos, tratando de aclarar todos los parámetros que intervienen en la transmisión de las imágenes. Se propone también la configuración óptima de estos parámetros para que el sistema completo funcione de la forma más eficiente posible.

Finalmente se proponen a los interesados en este trabajo, futuras posibles líneas de desarrollo y mejoras que se pueden llevar a cabo partiendo de éste como base.

## 9.2 PRUEBAS REALIZADAS

El tamaño de las imágenes que pueden transmitir a priori las aplicaciones desarrolladas puede oscilar entre 1 byte y 1.4 Mb. Esto se ha implementado de esta forma porque con este rango de valores se satisfacen ampliamente las necesidades para las que han sido implementadas estas aplicaciones. A pesar de ello, su puede incrementar este rango todo lo que se quiera sin más que aumentar el tamaño de los correspondientes buffers de recepción de la aplicación Cliente tanto como se quiera.

Por este motivo, se han tomado 13 imágenes para realizar las pruebas, cuyos tamaños varían desde 1 byte hasta 1292 Kb, además de las correspondientes imágenes que transmite la cámara.

Estos ensayos se han realizado empleando las aplicaciones con y sin compresión que emplean la tarjeta digitalizadora PC-COMP de Imaging Technology. Para no repetir explicaciones, diremos que los resultados que se obtienen con las aplicaciones implementadas para la tarjeta Matrox son prácticamente los mismos ya que el motor de transmisión coincide exactamente en ambas aplicaciones. Tan sólo difieren los resultados de la parte referente a la transmisión de la imagen capturada por la cámara. El motivo de esta diferencia es que la tarjeta de Imaging captura una

imagen en color de 24 bpp y por tanto de un tamaño aproximado de 1.3 Mb. En cambio la tarjeta Matrox captura imágenes en blanco y negro de pequeño tamaño. Pero para estimar un comportamiento aproximado para la transmisión de esta imagen capturada por la tarjeta Matrox, diremos que es un comportamiento similar al de transmitir desde disco una imagen de 200 Kb ó 300 Kb de tamaño.

### 9.2.1 PRUEBAS DE COMPRESIÓN DE IMÁGENES

En primer lugar procederemos a realizar las pruebas pertinentes al algoritmo de compresión que se ha empleado para las aplicaciones que incluyen esta característica. Recordemos en líneas generales que se trata del algoritmo de Huffman Adaptativo. Es un algoritmo de compresión sin pérdidas y basado en la repetición de caracteres dentro de un archivo de datos. Otra de las premisas fundamentales a la hora de elegir este algoritmo de compresión fue la velocidad funcionamiento a la hora de programarlo, de forma que introdujera la menor carga adicional posible (en los referente al tiempo) a las aplicaciones que no emplean esta compresión.

Las pruebas realizadas sobre las trece imágenes elegidas han dado como resultado:

| NÚMERO DE IMAGEN | TAMAÑO ORIGINAL | TAMAÑO COMPRIMIDO | PORCENTAJE DE COMPRESIÓN |
|---|---|---|---|
| IM. 1 | 1 KB | 1 KB | 0% |
| IM. 2 | 2 KB | 1 KB | 50% |
| IM. 3 | 3 KB | 2 KB | 33% |
| IM. 4 | 9 KB | 3 KB | 67% |
| IM. 5 | 10 KB | 7 KB | 30% |
| IM. 6 | 71 KB | 60 KB | 15% |
| IM. 7 | 110 KB | 95 KB | 14% |
| IM. 8 | 201 KB | 183 KB | 9% |
| IM. 9 | 301 KB | 178 KB | 41% |
| IM. 10 | 429 KB | 304 KB | 29% |
| IM. 11 | 612 KB | 485 KB | 21% |
| IM. 12 | 703 KB | 678 KB | 4% |
| IM. 13 | 1292 KB | 1128 KB | 13% |

De media se obtiene un porcentaje de compresión del **27%**, que para tratarse de un algoritmo de compresión sin pérdidas consideramos que se trata de una buena relación de compresión.

El factor principal del que depende esta relación de compresión es la repetición de caracteres dentro de una imagen. Esta faceta es bastante común en las imágenes para las que se va emplear este trabajo ya que es habitual que en estas encontremos grandes regiones que pertenecen al mismo elemento y que por tanto presentan alto

grado de coincidencia. Aunque hemos de dejar claro que a priori, a la hora de elegir este algoritmo, no se hizo ninguna suposición acerca de la información a transmitir.

Visto que se consigue una tasa de compresión elevada, podemos afirmar que se ha cumplido el objetivo fundamental que nos propusimos a la hora de incorporar la posibilidad de la compresión de la información y que era el de descargar la red en la medida de lo posible sin que el tiempo adicional de proceso fuera demasiado perjudicial para nuestras aplicaciones. En cuanto al tiempo adicional, en el apartado siguiente comprobaremos que es perfectamente asimilable.

A modo de ejemplo, y empleando la tasa de compresión media que hemos obtenido, diremos que si nos ponemos en la situación perfectamente real de transmitir una secuencia de 100 imágenes de 700 Kb cada una, empleando las aplicaciones con compresión, en vez de transmitir 70 Mb de información vamos a transmitir 51 Mb, lo que supone un ahorro muy considerable en el ancho de banda de la transmisión. Queda por tanto comprobado que con la introducción del algoritmo de compresión se produce una descarga significativa de la red sobre la que funcionen las aplicaciones.

### 9.2.2 TIEMPOS DE TRANSMISIÓN

El segundo parámetro que se ha medido en los ensayos ha sido que transcurre desde que el usuario envía la petición de la imagen hasta que esta está disponible en la visualización de la aplicación Cliente. Para realizar esta medición de tiempo se han empleado las funciones de temporización de las que dispone Visual C++.

Se han medido los tiempos para las aplicaciones sin compresión y con compresión. Además, para tener un concepto mejor formado del funcionamiento de la red, este estudio de tiempo se ha realizado entre dos equipos muy próximos dentro de esta red de área local y dos equipos muy distantes.

Los tiempos de transmisión van a depender de dos factores fundamentales:

- El tamaño de la imagen que se desea transmitir.
- El nivel de carga de la red.

La red de área local que se encuentra instalada en el edificio de laboratorios del Departamento de Ingeniería de Sistemas y Automática de la Escuela de Ingenieros de Sevilla, tal y como se describió en el apartado 2.4, se trata de una red Ethernet a

10Mbps montada en topología en bus y con un cableado de tipo UTP. Haciendo referencia al capítulo 3, podemos afirmar que aunque la configuración en que está instalada esta red, tanto en distribución como en cableado, está bastante extendida, en ningún momento es la más eficiente desde el punto de vista de la velocidad de transmisión de la información.

El número de equipos conectados a esta red es bastante elevado. Asimismo, esta red dispone de acceso a Internet a través de un servidor proxi. Estas dos circunstancias provocan que en todo momento el tráfico que soporta la red sea bastante elevado. Esto es una circunstancia adicional a la hora de interpretar los resultados que se incluyen a continuación.

El tiempo que vamos a medir no se reduce exclusivamente a la trasmisión de las imágenes, sino que vamos a incluir el tiempo de proceso necesario para llevar a cabo el acondicionamiento de la información y, para el caso con compresión, también los tiempos de compresión y descompresión. Por este motivo, el tiempo medido en los casos con compresión es superior, aunque debemos tener presente que el tiempo en que se ha estado utilizando ancho de banda de la red es significativamente menor.

Los resultados obtenidos para las mismas trece imágenes son:

| | SIN COMPRESIÓN | | CON COMPRESIÓN | |
|---|---|---|---|---|
| NÚMERO DE IMAGEN | EQUIPOS PRÓXIMOS | EQUIPOS DISTANTES | EQUIPOS PRÓXIMOS | EQUIPOS DISTANTES |
| IM. 1 | 91 ms | 290 ms | 100 ms | 452 ms |
| IM. 2 | 150 ms | 541 ms | 250 ms | 525 ms |
| IM. 3 | 110 ms | 361 ms | 260 ms | 421 ms |
| IM. 4 | 100 ms | 301 ms | 271 ms | 582 ms |
| IM. 5 | 180 ms | 350 ms | 280 ms | 662 ms |
| IM. 6 | 151 ms | 400 ms | 385 ms | 693 ms |
| IM. 7 | 220 ms | 471 ms | 650 ms | 955 ms |
| IM. 8 | 210 ms | 811 ms | 825 ms | 1355 ms |
| IM. 9 | 271 ms | 1041 ms | 1223 ms | 2444 ms |
| IM. 10 | 351 ms | 1312 ms | 1845 ms | 2955 ms |
| IM. 11 | 530 ms | 1603 ms | 2013 ms | 4321 ms |
| IM.12 | 440 ms | 1952 ms | 1650 ms | 6521 ms |
| IM.13 | 1062 ms | 3125 ms | 3200 ms | 7210 ms |

Vemos que el tiempo de proceso introduce un retraso, no en la transmisión, sino en el procesado de la imagen. Este aumento del tiempo en la representación de la imagen no es tiempo de transmisión propiamente dicho. Es decir, aplicaciones que utilicen el código de los programas sin necesidad de mostrar la imagen al usuario mejoran el tiempo efectivo del trabajo.

También es de vital importancia considerar en este punto que las pruebas se han realizado con el servidor de imágenes funcionando en el equipo de trabajo que se

describe en el capítulo 2 de esta memoria, que como recordemos posee un microprocesador *Intel Pentium II a 300 Mhz*. Es sabido que es un equipo de características bastante anticuadas con respecto a los equipos que se instalan hoy día, cuyos procesadores funcionan a velocidades próximas a los 3 Ghz. Esto indica que si instalamos nuestras aplicaciones en un PC de características más acordes a los tiempos en que vivimos, podemos reducir el tiempo de manera muy considerable. Así tendremos que el tiempo de funcionamiento de las aplicaciones con compresión es inferior al de las aplicaciones que no la emplean, con lo cual no tendremos que pagar ningún precio adicional a cambio del considerable ahorro de ancho de banda que supone para la red.

A pesar de esto, observando los tiempos de transmisión, y conociendo el entorno donde se van a emplear estos programas, todos los casos están dentro de valores razonablemente buenos de transmisión de datos a través de red Ethernet y el incremento de tiempo que introduce el proceso de compresión y descompresión es un costo más que razonable a pagar a cambio del importante ahorro que introducen las aplicaciones con respecto al nivel de carga de la red. Hemos de tener en cuenta que para el caso del proyecto de detección de humo, se toman imágenes de pequeño tamaño en intervalos que rondan los dos segundos. Por este motivo, todas las aplicaciones desarrolladas cumplen las restricciones de tiempo que podría exigir este sistema.

## 9.3 ENTORNO DE FUNCIONAMIENTO MEJORADO

Para obtener los mejores resultados posibles con estas aplicaciones, se deberían hacer las siguientes mejoras en el entorno donde se implanten:

- Utilizar equipos PC´s actuales para que se reduzca en medida de lo posible el tiempo de proceso.
- Modificar la topología de la red y reducir el número de equipos que la conforman. Aunque la topología en bus está muy extendida y es una solución bastante económica, esta resulta poco eficiente en la transmisión de los datos.
- Sustituir las tarjetas de red por otras de 100 Mbps de las que se dispone hoy día en el mercado.

- Utilizar cableado de fibra óptica o cable coaxial, que aunque más caro mejora de forma considerable los tiempos de transmisión.

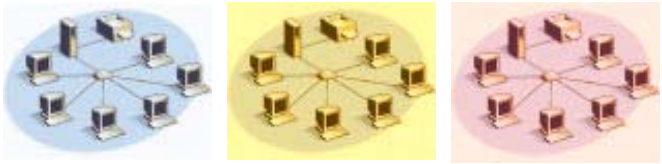
## 9.4 POSIBLES LINEAS DE INVESTIGACIÓN

El trabajo desarrollado durante la realización de este proyecto fin de carrera, se ha comprobado que cumple con las necesidades para las que fue diseñado. No por ello, este sistema deja de poder ser mejorable o incluso ampliable para ser implementado en otros entornos que requieran de mayor funcionalidad. Nadie mejor que la persona que ha estado desarrollando las aplicaciones para proponer las posibles líneas de desarrollo que se pueden llevar a buen término partiendo de este trabajo como base de estas futuras investigaciones. Estas son:

- Se podría ampliar el ámbito de aplicación de este proyecto si se desarrollaran las aplicaciones para trabajar con un mayor número de formatos gráficos, no solamente BMP.

- Se podrían introducir otros algoritmos de compresión sin pérdidas, para implantar este sistema en entornos donde no se prime la velocidad de funcionamiento y sí la descarga de la red.

- La tercera y última línea de investigación podría ser el desarrollo de aplicaciones multihilo para que el Servidor pueda atender peticiones de distintos Clientes simultáneamente o pueda tratar la recepción y la representación de los datos en paralelo.



# BIBLIOGRAFÍA

# BIBLIOGRAFÍA

Para terminar la memoria de este proyecto fin de carrera incluimos, como es preceptivo, la bibliografía.

La estructura de la que consta esta parte del proyecto está basada en el *"MANUAL DE ESTILO DE PUBLICACIÓN DE LA AMERICAN PSYCHOLOGICAL ASSOCIATION"* (*APA*).

# REFERENCIAS

## Libros:


- Fortier, Paul J. Handbook of LAN Technology. McGraw Hill.
- González, Rafael C. & Woods, Richard E. Tratamiento Digital de Imágenes. Addison Wesley/Diaz de Santos.
- Kruglinski, David G. Programación Avanzada en Visual C++. McGraw Hill.
- Meyer, Bertrand. Construcción de Software Orientado a Objetos. Prentice Hall.
- Microsoft. Fundamentos de Redes Plus. Curso Oficial de Certificación. McGraw Hill.
- Pressman, Roger S. Ingeniería del Software: Un Enfoque Práctico. McGraw Hill.
- Raya, José Luis. & Raya, Laura. Cómo Construir una Intranet con Windows 2000 Server. RA-MA.
- Stevens, Al. & Walnum, Clayton. Programación con C++. Anaya Multimedia.
- Stroustrup, Bjarne. El Lenguaje de Programación C++. Addison Wesley.


## Páginas Web:


- www.conclase.net
- www.codeguru.earthweb.com
- www.matrox.com
- www.imaging.com
- www.jai.dk
- www.microsoft.com