Capitulo 4 Implementación del Sistema.

La implementación del sistema consta de la integración de la PDA en una red inalámbrica, el desarrollo de los archivos XML para guardar los datos y la implementación del cliente y el servidor. También describiremos el funcionamiento de las clases más usadas en este sistema.

La implementación del sistema presentó algunos problemas. Por una parte el lenguaje de programación utilizado (C#) es relativamente nuevo, por lo que no hay suficiente información en línea cuando surgen dudas específicas. Las aplicaciones para PDA están empezando a tener auge entre los desarrolladores, pero los ejemplos y tutoriales para desarrollar las aplicaciones son escasos. Durante este capitulo, se explicarán algunos de los problemas encontrados durante la implementación, y la manera en que se resolvieron.

4.1 Red Inalámbrica.

La integración de nuestro sistema a una red inalámbrica no significó mucho problema, el PDA y el Router Hub se integran de manera directa a la red. Se utilizó una HP ipaq h5555 y un Router Hub “Netgear Wireless Router MR814V2”. Con el Router, la asignación de direcciones IP ser hace de manera automática y sin contratiempos. Se le asigna una IP estática a la PC que funcionará como servidor, para que el cliente se pueda conectar desde la PDA.

La PC utilizada es una computadora Intel Pentium 4 con Windows XP. La PC consta de una tarjeta de red Ethernet de 10/100 Mb por segundo, la conexión con el router es por medio de un cable de red, es decir no es inalámbrica. La PDA cuenta con una tarjeta de red inalámbrica interna. Utiliza el protocolo inalámbrico 802.11b, y opera

a una velocidad de 11 Mb por segundo, tiene un radio de conexión de 5 metros, mientras que el router tiene un radio de conexión de 30 metros.

Las pruebas se realizaron conectando la PDA a Internet mediante el router, y una conexión a Internet por medio de un cable MODEM. También se probó la conexión con la PC "servidor", accesando carpetas compartidas de la red interna.

4.2 Implementación de las acciones del sistema (servidor).

El servidor como ya lo mencionamos anteriormente está integrado con el generador de histogramas y anotaciones. Al implementar el sistema, lo primero que se desarrolló fue el generador de histogramas, ya que del desarrollo de este, dependería el éxito de nuestro sistema. Después se desarrolló el generador de anotaciones ontológicas, y la manera de escribir archivos XML. Finalmente, se desarrollaron las aplicaciones de Servidor para realizar las consultas.

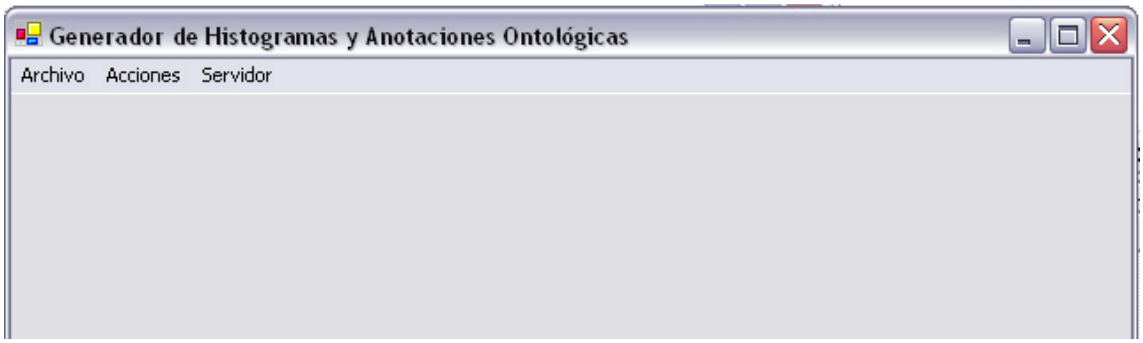

Figura 4.1 Menus Principales del Sistema.

Como podemos observar en la figura 4.1, se utilizó una aplicación de ventana utilizando formas de Windows en C#. Los menus principales son los de Archivo, Acciones y Servidor. Más adelante se irán explicando como funcionan.

La aplicación permite desde el menú archivo abrir una imagen del disco duro, para después ser dada de alta dentro del menú Acciones. El menú archivo también

permite guardar los histogramas, y anotaciones de todas las imágenes que se hayan dado de alta en la sesión actual. Esto optimiza memoria, al escribir a archivo los datos cada vez que el usuario lo desee, y no cada vez que una imagen se haya dado de alta.

Al elegir la opción de Abrir Archivo, se puede buscar en el disco duro la imagen que se desee dar de alta. Una vez que la imagen esté abierta en el programa, podemos darla de alta. En la figura 4.2, podemos observar la imagen en el fondo y la forma de llenado de Anotaciones Ontológicas. Después de haber llenado la forma, el sistema automáticamente genera el histograma de la imagen, convirtiéndola a escala de grises, y volviéndola a dibujar en nuestra aplicación como se muestra en la figura 4.3

Figura 4.2 Forma de Anotaciones.

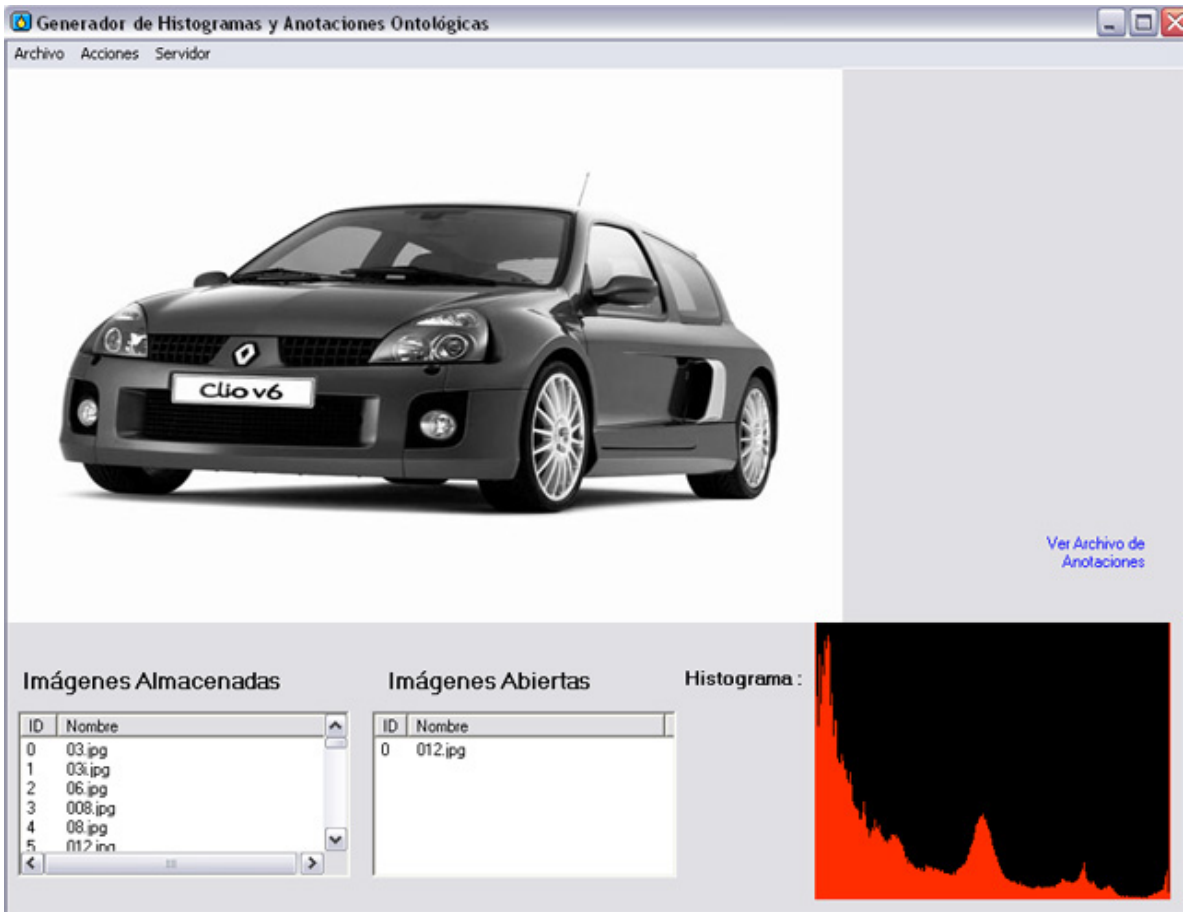

Figura 4.3 Imagen Convertida a Escala de Grises

Además podemos observar en la Figura 4.3 que se genera el histograma de escala de grises así como el nombre del archivo de la imagen que dimos de alta. Al guardar las imágenes, la lista de imágenes se anexará a la de Imágenes almacenadas.

Al volver a abrir un nuevo archivo, éste tomara el lugar de la imagen que se esté mostrando en el momento. De esta manera, podemos estar abriendo imágenes que estaremos visualizando, mientras las damos de alta. De igual forma, podemos abrir una imagen sin tener que darla de alta, esto es en el caso de haber seleccionado una imagen por error, el visualizarla en la aplicación nos permite asegurarnos de haber abierto el archivo correcto. Bien pudimos haber automatizado el proceso, permitiéndole al usuario abrir varias imágenes a la vez, para obtener de manera automática el histograma de grises de todas y cada una de las imágenes. El problema es que las anotaciones

ontológicas deben ser provistas por el usuario, de manera individual con cierta dedicación y paciencia, cuidando de que los datos se ingresen de la manera más homogénea posible.

Dentro del menú de Servidor, tenemos la opción de comenzar en donde el sistema genera una nueva ventana para nuestro servidor, como se muestra en la figura 4.4

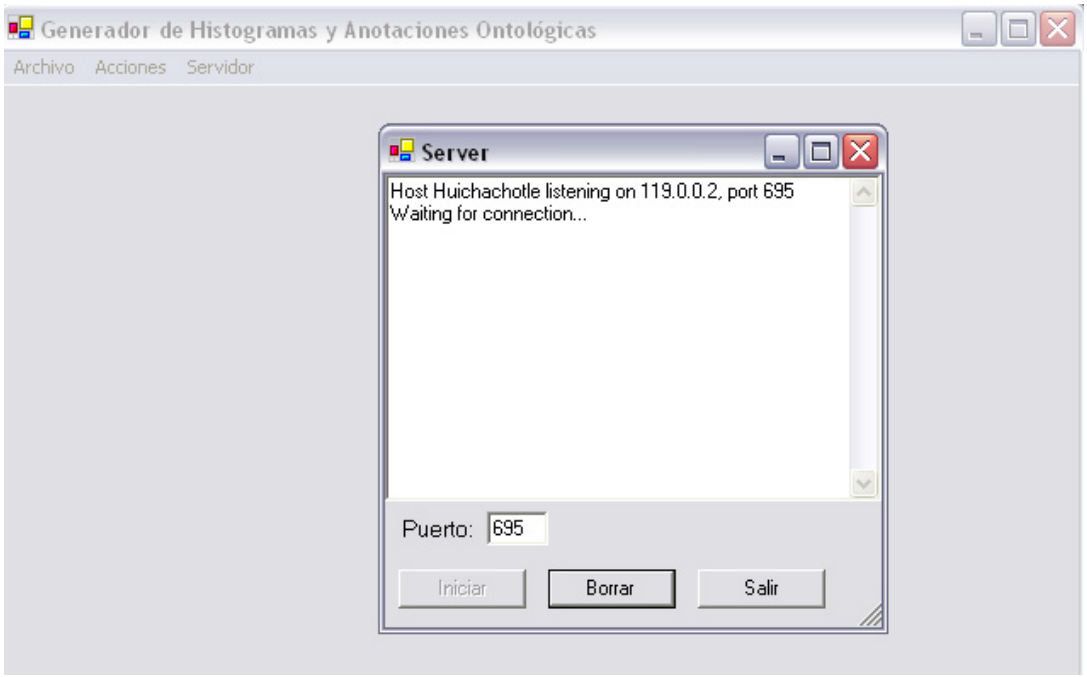

Figura 4.4 Interfaz del Servidor.

Al cargar la ventana del servidor, tenemos la opción de Iniciar el servidor, lo cual comienza el proceso de espera a una conexión del cliente. Dentro del campo de texto, se van generando los mensajes sobre el estado del servidor. Es decir, si se conectó algún cliente; si recibió una consulta y si logró satisfacerla. El botón de borrar permite al usuario eliminar todos los mensajes, para no saturar el campo de texto. Finalmente el botón de salir termina las conexiones y cierra la ventana. El botón de salir solo se encuentra habilitado mientras no se esté realizando ninguna consulta. Esto es para no dejar incompleto el proceso de escritura de algún archivo.

El campo de texto de puerto puede ser cambiado manualmente, antes de iniciar el servidor, esto le da la opción al usuario, de elegir el puerto que sea más conveniente para que se ejecute el servidor.

Durante el desarrollo del sistema, se generaron algunas funciones extras para probar los resultados del sistema, ya que en las primeras etapas del desarrollo del sistema no se contaba con el cliente/servidor. Entre ellas, hay una función que nos escribe a un archivo de texto, todas las imágenes existentes en nuestra colección. En el capitulo de Pruebas se explicarán las funciones adicionales que se utilizaron para realizar las pruebas de nuestro sistema.

4.3 Implementación de las Acciones del Sistema (cliente).

El cliente de nuestro sistema requirió de más investigación sobre como desarrollarlo. El sistema Pocket PC aunque comparte muchas librerías con Windows, estás son muy limitadas. Cuando se implementó el software del cliente, se pensó en que antes de poder realizar cualquier consulta, el sistema debía verificar si la dirección IP era válida.

En la figura 4.5, podemos observar la imagen principal que se despliega al cargar nuestro programa. La opción de Menú está deshabilitada. Así, el usuario debe ingresar primero la dirección IP, y dar clic al botón de conectar. Una vez que se comprueba que existe un servidor, se habilita el Menú, y nos permite realizar las consultas.

Lo primero que se planteo, fue la manera en que se realizarían las vistas de los resultados de las consultas. Como iba a ser muy tedioso el estar enviando las imágenes del cliente al servidor con todos los datos, se optó por generar diferentes “vistas” de los archivos XML que serían cargados por el cliente de manera remota.

Figura 4.5 Interfaz del Cliente

Figura 4.6 Consulta del Cliente.

Al iniciar el sistema, el cliente carga de manera remota un archivo XML predefinido, que contiene las primeras imágenes. Para transformar este archivo, utilizamos un XSL predefinido que nos genera código HTML.

En la figura 4.6, se muestra la manera en que se despliega el archivo XML ya transformado con el archivo XSL. En la parte de arriba, se pueden modificar el número de resultados que queremos obtener por la siguiente consulta, el numero de campos en las anotaciones ontológicas que queremos que sean iguales, y si queremos o no, incluir la búsqueda por color (ésta se encuentra activada por default). En la parte de abajo está el botón de Buscar, y al lado podemos elegir el número de la imagen que queremos utilizar en la consulta.

Cada vez que realizamos una consulta, el servidor genera un archivo XML con todos los resultados obtenidos en la búsqueda. El cliente carga el nuevo archivo, y éste se despliega para el usuario. Si el resultado de la consulta incluye una Búsqueda por histogramas, en el archivo XML se incluyen la distancia euclideana de cada imagen con respecto a la elegida en la consulta. En general, el Cliente permite la visualización de las consultas de una manera clara, desplegando tanto las imágenes como sus anotaciones ontológicas. De no querer incluir la búsqueda por sus descripciones, se deja en 0 el valor de # Onto. Aún cuando no se incluya la búsqueda por anotaciones ontológicas, estas se despliegan junto con el archivo de resultados. Esto es para permitir al usuario conocer los datos completos de la imagen.

4.4 Implementación del Procesamiento de la Imagen.

El procesamiento de la imagen se realiza desde la clase Histogram, y se guarda junto con los histogramas, en un archivo serializado.

Dentro de la interfaz principal, cargamos un archivo con un OpenFileDialog, que nos permite buscar un archivo dentro de nuestro disco duro. En un principio, se pensó poder dejar las imágenes en cualquier parte del disco, y guardar la ruta completa del archivo, pero esto solo generaría más problemas, porque nuestra colección de imágenes

estaría esparcida por todo el disco duro. Se decidió entonces definir una carpeta, donde se guardarían tanto la colección de imágenes, como los archivos XML y XSL.

La imagen se procesa con el método Get Histogram de la clase Histogram. Get Histogram recibe la imagen en Bitmap, y una estructura de tipo Imagenes. La primera tarea de Get Histogram, es convertir a escala de grises cada píxel. Se toman los valores RGB de cada píxel, y cada valor se multiplica por un valor decimal.

```
for(int y=0;y<b.Height;y++)
{
	for(int x=0; x < b.Width; x++ )
		{
			blue = p[0];
			green = p[1];
			red = p[2];

		p[0]=p[1]=p[2]= (byte)(.299 *red+.587 *green+.114 *blue);
			if(p[0]<0)
				imagen.histograma[0]++;
			else

				imagen.histograma[p[0]]++;
			p += 3;

		}
	p += nOffset;
}
```

Como podemos observar en el código, se multiplica un valor diferente por cada color, para convertirlo a escala de grises. Esta formula permite que la tonalidad de gris de un píxel, no sea la misma al tener valores invertidos en alguno de sus colores. Con el método convencional para convertir a escala de grises (R+G+B)/3, si tuviéramos un píxel con un valor R=0 , G=1 y B=1, y otro con los valores R=1, G=0 y B=1, al convertirlo a tonos de grises, ambos píxeles serían iguales.

Dentro de imagen.histograma hay 256 posiciones. Esto es por que solo se generan 256 tonos de grises posibles. Por eso se incrementa la posición con el valor de

p[0], que sería el tono que se generó al convertirse en escala de grises. De esta manera, se van incrementando los valores de cada posición, hasta terminar de procesar la imagen completa. Al final, habremos obtenido el histograma en un arreglo de 256 enteros.

Una vez que la imagen ha sido procesada, el histograma está completo. Get Histogram recibe la Estructura de tipo Imagenes como referencia, por lo que solamente regresa un “true”. Una vez que se elige la opción de guardar los Histogramas, el programa serializa el arreglo de Imagenes para guardarlo en archivo. El número identificador de cada histograma, es la posición del arreglo en donde se encuentra, y también se guarda en el archivo XML, con las anotaciones ontológicas. El sistema utiliza el método EscribeArchivo de la Clase Archivos, para serializar el arreglo de Imágenes, y guardarlo en el archivo Imágenes.txt.

La razón para serializar los archivos es que, permite guardar mayor información en menos espacio. Esto también agiliza el proceso de guardado. El único problema es que al abrir un archivo serializado, con un procesador de palabras, el contenido es ilegible. Cada que el sistema se inicia, se acceso el archivo serializado, y se lee su contenido que se guarda en un arreglo de Imágenes. Esto permite tener siempre la información disponible para realizar consultas, o dar de alta nuevas imágenes. Al no estar abriendo y cerrando el archivo, cada que se requiera leer o agregar datos, se reducen los riesgos de perder información.

```
public static void EscribeArchivo(Imagenes[] imagenes, int i)
        {
                Stream outputStream=
                File.OpenWrite(@"C:\imagenes.txt");

                BinaryFormatter bf = new BinaryFormatter();
                bf.Serialize(outputStream,i);
                bf.Serialize(outputStream, imagenes);
                outputStream.Close();
        }
```

En el código anterior, podemos ver la forma en que trabajan los archivos serializados en C#, cabe mencionar que todo código que se planee serializar debe anteponerse [Serializable] para que el compilador sepa que el código siguiente será serializado después. En este caso se realizó al crear la estructura Imágenes.

4.5 Implementación de las Anotaciones Ontológicas.

Para definir la estructura de las anotaciones ontológicas, se recurrió al uso de archivos XML ya que son precisos, fáciles de manipular, y fáciles de entender. Un archivo XML mantiene una jerarquía de árbol, tiene la ventaja de que puede ser interpretado por cualquier navegador de Internet reciente, y al abrirlo con un procesador de texto, un usuario puede entenderlo y modificarlo fácilmente.

El árbol de anotaciones que se utilizó se puede ver ilustrado en la figura 4.7. Como podemos observar, se tiene la raíz que es medio de transporte, y cada uno de sus nodos es un "Archivo", que a su vez tiene varios nodos que guardan la descripción de la imagen.

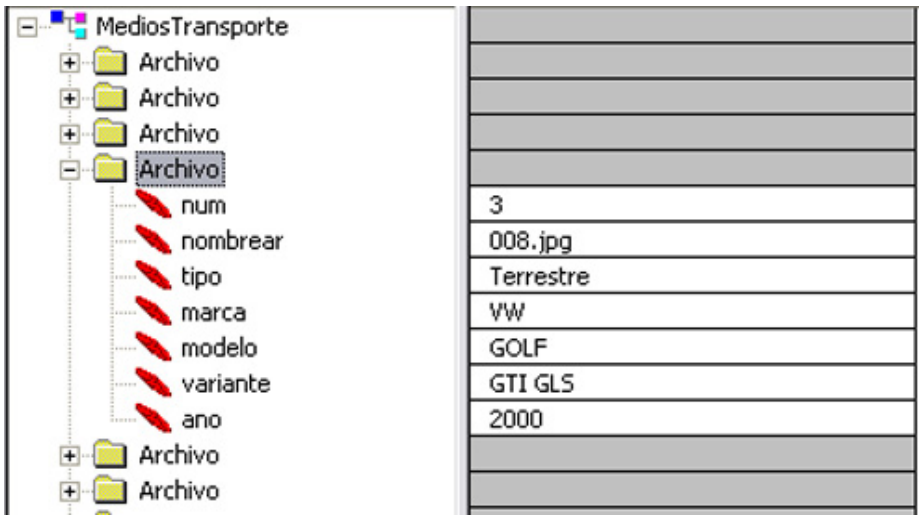

Figura 4.7 Árbol de Anotaciones.

El esquema básico utilizado para generar nuestro archivo XML raíz fue el siguiente:

```
<?xml version="1.0"?>
<MediosTransporte>
  <Archivo>
    <num>0</num>
    <nombrear>03.jpg</nombrear>
    <tipo>Terrestre</tipo>
    <marca>VW</marca>
    <modelo>GOLF</modelo>
    <variante>GTI 1.8T</variante>
    <ano>2000</ano>
  </Archivo>
</MediosTransporte>
```

Donde <num> se refiere al número identificador del archivo. De esta manera, mediante nuestra forma de anotaciones ontológicas, podemos ir agregando los campos de <Archivo> con el contenido escrito por el usuario. Se decidió no encerrar ningún campo de otro, para no limitar nuestros resultados, ya que pueden existir dos medios de transporte de la misma marca, de diferentes tipos (ej. Una lancha y un Auto de marca Zuzuki).

Los datos obtenidos mediante la forma de anotaciones ontológicas, se guardan en un arreglo de estructuras de tipo OntoTemp. Esta estructura es por decirlo de una manera un <Archivo>, ya que guarda todos los datos que se encuentran dentro de este elemento del archivo XML.

Cuando el usuario elige la opción de “Guardar”, se llama al método GetXml() de la clase Archivos. Esta clase se encarga de abrir el archivo “Ontologia.xml” que guarda todas las anotaciones ontológicas de la colección de imágenes que se hayan dado de alta. El archivo es leído, y después se agrega el contenido del arreglo de estructuras de tipo OntoTemp. Después se guarda nuevamente el archivo.

La ventaja de utilizar un archivo XML para guardar las anotaciones ontológicas, es que el administrador del sistema puede abrirlo fácilmente, y corregir posibles errores o actualizar información, todo desde cualquier procesador de palabras. Cabe mencionar que no se pueden dar de alta nuevas imágenes, ni modificar los números identificadores de cada imagen.

Para las vistas de las consultas, se generaron 3 tipos de archivos XSL, y 2 tipos de archivos XML, todos muy similares al presentado anteriormente, pero con algunos campos extras.

Se agregó el elemento <distancia> que contiene la distancia euclideana del archivo para las consultas que lo requieran. Se creo un campo <id>, en donde se encuentra el número de resultado generado en el esa consulta. Por ejemplo, si el usuario en su consulta solicitó que se le regresaran 12 imágenes similares, dentro del archivo de resultados, cada <id> tendrá un valor del 1 al 12, donde 1 es el archivo más similar al elegido en la consulta, y 12 el archivo menos similar. Se agregó también la referencia al archivo XSL, con el que debería de ser abierto cada XML.

Los archivos XSL tienen la ventaja de que son llamados por los archivos XML. Al abrir un archivo XML, de tener éste una referencia a un archivo XSL, automáticamente se abre el XSL, y se genera el código HTML pertinente.

```
<?xml version="1.0"?>
<?xml-stylesheet type='text/xsl' href='resultadonto.xsl'?>
```

Como podemos observar en el código anterior, se está haciendo referencia al archivo “resultadonto.xsl”. Dependiendo de la que se realice, el sistema hace referencia al archivo correspondiente dentro del XML.

Dentro de los archivos XML de resultados se incluyo también un atributo al elemento raíz.

<tempo onto="YES">

El atributo “onto” informa si el archivo contiene anotaciones ontológicas. Esto es para consultas donde solamente se buscó por color (distancia Euclideana), y no se guardaron las anotaciones ontológicas.

4.6 Implementación del Cliente/Servidor.

La implementación del Cliente/Servidor resultó un tanto complicada. En un principio, se planeó utilizar un “wrapper” para enviar arreglos de estructuras a través del servidor al cliente. Un “wrapper” envuelve un objeto para poder ser enviado a través de un protocolo TCP/IP. El problema es que las librerías del PDA Pocket PC todavía no soportan “wrappers”, lo único que nos permiten enviar a través de una conexión a red son cadenas binarias.

Este problema retraso mucho la implementación del sistema, porque vino a modificar muchos procesos que inicialmente se habían creado en el diseño. También se tuvo que modificar la arquitectura original del sistema, ya que se tenía planeado realizar algunos procesos dentro del cliente.

La solución de este problema se obtuvo enviando solamente la consulta mediante un solo “string”, que se convertiría a una cadena binaria para ser recibido por el servidor. Se utilizó un “string”, ya que este es un arreglo de caracteres que puede contener una especificación por cada valor de la cadena.

El cliente captura la información obtenida de la forma de consulta. Cada valor entero es convertido a un carácter, y guardado en un una posición de la cadena de caracteres. Después se convierte la cadena de caracteres en un Stream Binario, y se

envía al servidor. El servidor recibe el stream binario, lo convierte nuevamente a una cadena de caracteres, y después convierte a enteros cada posición de la cadena. Dependiendo de los valores obtenidos, se realizan los diferentes tipos de búsquedas, y se escriben los diferentes tipos de archivos.

El servidor es el encargado de llamar a todos los métodos de búsqueda, se encarga de juntar los resultados, y de llamar los métodos que escriban los resultados en un archivo XML.

Por su parte, el cliente se encarga de capturar los datos obtenidos del usuario referentes a las consultas, y de mostrar el contenido de los archivos XML, para que el usuario pueda visualizar las imágenes y su contenido ontológico. El cliente válida que el servidor realmente exista antes de realizar cualquier consulta.

4.7 Implementación de las Búsquedas.

Las búsquedas son manipuladas por el servidor ya que es la clase que recibe los parámetros de la consulta, la que se encarga de generar los archivos XML con los resultados, y la que envía la respuesta al cliente.

4.7.1 Busqueda OntoId.

La primera búsqueda que se realiza, es la de la imagen elegida por el usuario dentro del archivo XML de resultados actual. De ser la primera consulta, la búsqueda se hace sobre el archivo XML predefinido. Como ya mencionamos anteriormente el cliente solo envía una cadena de caracteres para especificar la consulta. Por lo tanto, el primer dato que toma el servidor es el <id> del archivo elegido por el usuario. Este id es el número identificador dentro del archivo XML de resultados, y es muy diferente de <num> que es el número identificador de un archivo en la colección de imágenes del

sistema. A diferencia de <num>, <id> cambia cada vez que se genera un archivo XML de resultados.

La primera búsqueda utiliza el método OntoId de la clase Busqueda para guardar las anotaciones ontológicas del archivo de resultados para poder realizar la consulta. Los datos se guardan en una estructura de tipo OntoTemp. El elemento <num> dentro del archivo XML sirve para realizar las búsquedas tanto de histogramas como de anotaciones ontológicas. OntoId regresará la estructura de tipo OntoId con los datos obtenidos del archivo XML.

4.7.2 Busqueda OntoXml.

OntoXml recibe la estructura obtenida en OntoId y la profundidad para buscar las similitudes entre las anotaciones ontológicas. Los archivos de búsquedas de XML son similares, por lo que usaremos el ejemplo siguiente para explicar un poco el funcionamiento del código.

```
for (int i=0; i<root.ChildNodes.Count; i++)
        {

                reader.ReadStartElement("Archivo");
                reader.ReadStartElement("num");
                num= Int32.Parse(reader.ReadString());
                reader.ReadEndElement();
                reader.ReadStartElement("nombrear");
                a=reader.ReadString();
                if(a==consulta.a)
                cont++;
                reader.ReadEndElement();
                reader.ReadStartElement("marca");
                b=reader.ReadString();
                if(b==consulta.b)
                cont++;
                reader.ReadEndElement();
//Continua…….
```

Como podemos ver en el código anterior, se utiliza un “for” para ir recorriendo los elementos <Archivo>. Cada “if” incrementa un contador que nos marca el número de elementos que tienen el mismo contenido que la imagen a consultar. Por ejemplo, si el contenido de “marca” es igual al campo de marca de nuestra imagen a consultar (consulta.b), entonces “cont” se incrementa en una unidad, y su valor máximo es el número de campos en la anotación ontológica.

```
if((cont>=profundidad)&&(a!=consulta.a))
        {
                resultado[nom]= new OntoTemp(a,b,c,d,e,f);
                resultado[nom].numero=num;
                nom++;
        }
```

Al final de leer cada elemento <Archivo>, se checa si el archivo cumple con el número de coincidencias solicitadas por el usuario, y si no tiene el mismo <num> (si no es el mismo archivo). Si estas condiciones se cumplen, entonces se guarda en un arreglo de estructuras OntoTemp llamado resultado. Se incrementa entonces el valor de “nom” que es el número de resultados obtenidos hasta el momento. Como podemos ver, la búsqueda por Anotaciones Ontológicas se hace de manera directa sobre el archivo XML, esto puede tomar más tiempo que si lo hubiéramos hecho leyendo un arreglo de estructuras, pero como ya lo mencionamos anteriormente, el hecho de tener los datos guardados en un archivo XML, facilita el entendimiento de la estructura de nuestras anotaciones para cualquier usuario.

El método OntoXml nos regresa un arreglo de estructuras OntoTemp. Este arreglo es el que se envía posteriormente, a nuestro método que genera nuestro archivo XML de respuestas.

4.7.2 Busqueda DistanciaEu.

Esta búsqueda recibe 2 valores, la estructura de tipo Imágenes con la imagen que se eligió, la colección de imágenes en un arreglo de estructuras de tipo Imágenes, y el número de imágenes de la colección lo recibe en un valor entero.

La búsqueda genera las distancias Euclideanas entre la imagen elegida y el resto de imágenes de la colección. El proceso se describió en el capitulo anterior, pero aquí pondremos el código final que se utilizó

```
for (int j=0;j<i;j++)
{
        tempdist=0;
        for(int k=0;k<256;k++)
        {
                tempdist2=0;
                tempdist2=(double)(imagen.histograma[k]-
                                        imagenes[j].histograma[k]);
                tempdist2=tempdist2*tempdist2;
                tempdist = tempdist+tempdist2;
        }
        tempdist=Math.Sqrt(tempdist);
        distancias[j]=new
                        Distancia((long)tempdist,imagenes[j].nombre,j); }
```

En el código anterior podemos observar como quedó nuestra función inicial planteada en el diseño del sistema, y ahora completamente funcional. Dentro de “tempdist2” vamos guardando la diferencia entre distancias Euclideanas de los píxeles de cada histograma. Después se guarda en un arreglo de estructuras Distancia. Al final, el archivo con la distancia más cercana al valor 0, es la imagen con un histograma de color más parecido a la imagen elegida en la consulta.

En realidad, esta consulta saca la distancia euclideana de todas las imágenes con respecto a la que solicitó el usuario. Una vez que se tiene todos los datos, se ordenan mediante un método de quicksort adaptado para ordenar arreglos de estructuras de tipo Distancias. Dependiendo del número de resultados que el cliente haya solicitado, es el número de <Archivos> que se escriben en el XML de resultados.

4.8 Escritura de Archivos.

La implementación de la escritura de Archivos se encuentra en una clase independiente, pero se utiliza en prácticamente todas las implementaciones, es por eso que se ha ido explicando sus aplicaciones conforme ha sido necesario. Los métodos se dividieron en: Métodos que escriben a archivos XML, métodos que escriben a archivos de texto (principalmente para pruebas), y el método que escribe a un archivo serializado. Los métodos pueden recibir estructuras de tipo Imágenes, OntoTemp o ambas según se requiera, de acuerdo a la consulta realizada por el cliente.

4.9 Conclusiones del Capitulo.

A lo largo de este capitulo se describió, en rasgos generales, la implementación del sistema. El principal reto fue el aprendizaje de un nuevo lenguaje de programación como loes C#, que no obstante que es muy parecido a Java, tiene muchas diferencias en la sintaxis y el modo en que funcionan sus librerías. Durante el desarrollo del sistema surgieron muchos problemas, generados principalmente porque al ser C# un lenguaje relativamente nuevo, todavía no hay suficientes documentos de referencia en línea, y aún hay menos impresos, sobretodo para la plataforma Pocket PC para la PDA, en donde el Compact Media Framework (las librerías de C# para programar aplicaciones en Pocket PC) se encuentra apenas en su primera versión, y todavía no se pueden integrar muchos de los componentes de C#.

Podemos ver que el uso de archivos XML permite integrar de una mejor manera las aplicaciones, permitiéndole no solo al desarrollador, sino al usuario final, tener un mayor entendimiento sobre el funcionamiento del sistema. Aunque su lectura puede ser mas lenta a comparación con un archivo serializado, los beneficios son los suficientes

como para justificar el tiempo extra que el sistema se tarda en cargar, y escribir el archivo.

Las fallas, y algunos errores, se mencionarán con mayor detalle en el siguiente capitulo. Podemos concluir que, la implementación del sistema, nos preparó para los detalles y errores que se presentarían durante la parte de pruebas de nuestro sistema.