Control remoto (web)
de un robot Aibo

Autor: José Manuel Cañete Poyatos

Director de proyecto: Jordi Vitrià Marca

Bellaterra, 12 de junio de 2006

Índice

1. Introducción

Este proyecto está contenido dentro de otro proyecto mayor (**proyecto global**) en el que se quiere que un conjunto de robots Aibo estén, durante 7 meses, en una exposición en la que se comportarán de manera autónoma, siendo capaces de identificar señales y adaptando su comportamiento conforme a esa identificación. La idea del proyecto global es que los robots sean tan autónomos como para que su comportamiento incluya el ser capaces de buscar (y encontrar) su cargador de batería y recargarse sin ninguna intervención humana.

En este proyecto se quiere montar un servidor de streaming que tenga una cámara que esté enfocada hacia los Aibos y que permita que los usuarios de Internet puedan ver lo que los robots van haciendo en la exposición, montada en el Palau Robert. Además de verlos, el usuario ha de poder controlar sus movimientos más básicos (adelante, atrás, girar a la derecha o a la izquierda), y también podrá ver lo que capte la cámara subjetiva de cada uno de ellos.

La primera fase es encontrar un programa que sea capaz de transmitir streams. Lo más corriente es que para servir un stream de contenido en directo se tengan que utilizar dos programas: uno que codifica lo que capta la cámara y que transmite lo codificado a otro programa, que es el que envía el stream a los clientes de Internet.

El streaming es transmisión de vídeo a través de una internet (por ejemplo, la Internet global) hacia ordenadores receptores, que pueden ver el vídeo conforme les van llegando los bits, sin la necesidad de esperar a que el fichero de vídeo se haya transmitido completo antes de que comience su visualización.

El vídeo transmitido puede ser de algo previamente grabado, o por el contrario puede ser un evento en directo, como es el caso que nos ocupa.

Para hacer streaming de ficheros ya creados, no es necesario un servidor especial donde colocar archivos de audio o vídeo. Un servidor web normal puede mandar la información y es el cliente el que se encarga de procesarla para poder mostrarla a medida que la va recibiendo. Sin embargo, para contenido en directo, como es nuestro caso, sí es necesario o bien un servidor de streaming o bien que el mismo servidor web haga también de servidor de streaming.

Para montar un servidor de streaming, hay que considerar diferentes puntos:

1. Ancho de banda, que determinará el formato de streaming, por tanto la elección de hardware y software. En este apartado hay 3 puntos a tener en cuenta:

    1.1. Ancho de banda de subida de los servidores para lanzar el stream hacia la red.

    1.2. Ancho de banda de la red que conecta (hace de puente de) la red del servidor con las redes de los clientes. En este caso, la red que hace de puente será Internet.

    1.3. Ancho de banda de bajada desde el punto de vista de los clientes.

    Para hacer streaming a través de Internet, para los clientes que usen módem de 56Kbps será más eficiente un stream a 36Kbps, y para clientes con ADSL o cable, puede ser eficiente desde 256Kbps a 1Mbps, dependiendo qué tipo de conexión tenga el cliente. Será, por tanto, deseable poder hacer streaming a diferentes anchos de banda para optimizar la calidad del vídeo del stream según el entorno de los clientes.

2. Selección del formato de streaming. Cada formato de streaming tiene su CODEC, y será más eficiente en uno u otro rango de ancho de banda. Por ejemplo, los formatos Windows Media 7 y 8, Windows MPEG-4, Real y Quicktime operan mejor en el rango de los 36Kbps a los 500Kbps. Windows Media 9 y MPEG4 lo hacen en el rango de los 500Kbps a los 3Mbps; MPEG-1 en el de 1Mbps a 3Mbps, y MPEG-2 de 3Mbps a 15Mbps.
Los codecs han de ser los mismos en el servidor y en los clientes.

3. El servidor puede utilizar *tarjetas de captura de vídeo*, que capturan, digitalizan y comprimen el vídeo y el audio de cámaras de vídeo, micrófonos u otros dispositivos. La digitalización y la compresión la llevan a cabo mediante los codecs. Estas tarjetas son, normalmente, tarjetas PCI

normales, y la selección de una u otra depende del ancho de banda al que se quiere realizar el streaming y del códec a utilizar, pero en nuestro caso no usaremos ninguna.
También podría dedicarse un ordenador con tarjeta de captura de vídeo exclusivamente para hacer de servidor de streaming, pero no será tampoco nuestro caso.

4. También hay que considerar el *software de streaming* a utilizar en el servidor y si se utilizará para hacer el streaming en *unicast* (es decir, desde el servidor a cada cliente uno por uno) o *multicast* (desde el servidor se envía una sola vez para que llegue a todos los clientes que estén conectados).

La manera más simple de servir streams es un *fichero* en *unicast*, ya que de esta forma no se necesita ningún software especial de streaming. Lo único que se necesita es codificar el vídeo en el formato y ancho de banda apropiados y crear los enlaces apropiados en la página web para que el usuario vea el stream al pulsar sobre ellos, momento en el que el reproductor instalado en el ordenador del cliente visualizará el vídeo, pero éste no es nuestro caso, ya que no queremos hacer streaming de un fichero, sino de lo que es captado por una cámara, sin crear previamente ningún fichero, transmitiéndolo en directo.

Esto se puede hacer, por ejemplo, con *unicast* en *directo*. También se puede con *multicast* para hacer llegar el stream a todos los miembros del grupo, es decir, se asignaría una dirección IP como grupo multicast y el servidor enviaría a esa dirección (dirección que empieza por 224.xxx.xxx.xxx) solamente un stream; los routers de la red se encargarían de duplicar y propagar el contenido, de manera que sólo se necesitaría una copia del contenido en cada rama de la red. Los clientes que quisieran ver el evento en directo serían miembros de ese grupo y tendrían la misma dirección IP multicast, e irían recibiendo los datos.

La ventaja en multicast es que se conserva el ancho de banda de la transmisión sea cual sea el número de usuarios que se añadan al grupo, por lo que solamente habría que reservar ese ancho de banda en cada rama de la red para el stream.

Sin embargo, en Internet no está asegurado el reencaminamiento multicast: los miembros de un grupo multicast pueden estar en redes diferentes, y en ese caso se necesitan routers especiales que sean capaces de reencaminar la información. Si un usuario quiere ser miembro de un grupo y está en una red que sólo está conectada con otras redes a través de un router, si ese router no es capaz de hacer multicast, ese usuario no podrá unirse al grupo.

Respecto a la cámara subjetiva de un Aibo, lo deseable sería poder usar la misma tecnología que para el stream de contenido en directo.

Por otro lado, ha de incluirse la posibilidad de controlar los Aibos a través de Internet, pero teniendo en cuenta el limitado número de Aibos, será necesario disponer de un mecanismo que deje solamente controlar un Aibo a un usuario a la vez, de forma que no haya lucha por el control del Aibo. Cada usuario que quiera manejar un mismo Aibo, deberá pedir turno.

2. Fases de desarrollo

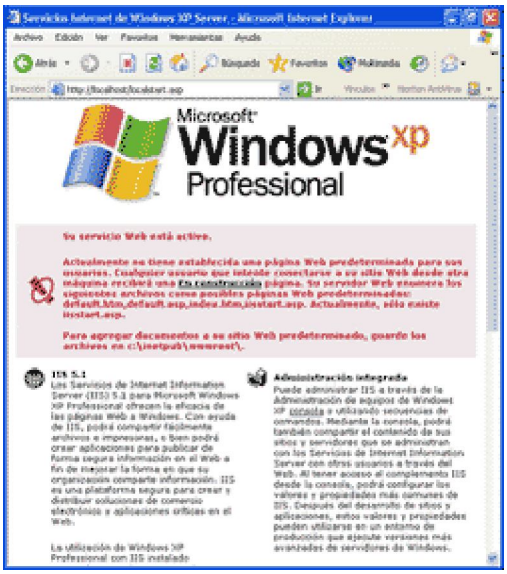
2.1. Instalación de los elementos necesarios en el servidor

Elección del sistema operativo: Windows XP Profesional (por estar ya instalado en los ordenadores del laboratorio).

2.1.1. Instalación del servidor web Internet Information Services (IIS):

En panel de control -> Agregar o quitar programas -> Agregar o quitar componentes de Windows, marcar Servicios de Internet Information Server (IIS) y en detalles marcar Archivos comunes, Complemento de Servicios de Internet Information Server (para poder configurar fácilmente IIS), la documentación y el Servicio World Wide Web. Dentro de éste último sólo es necesario marcar la opción Servicio World Wide Web.

Para comprobar si se ha instalado correctamente IIS escribimos http://localhost en Internet Explorer y debe aparecer una página web informando que IIS está correctamente instalado. Además, aparecerá en una ventana emergente la documentación de IIS instalada.



Aspecto de la página que ha de aparecer en el navegador

2.1.2. Instalación de php

En el ordenador del laboratorio, lo instalamos en **D:\php.**

Hay 2 maneras de instalarlo: manual o con instalador. La opción escogida ha sido manual, ya que no entraña complicación. Sólo hay que descomprimir el fichero descargado del sitio oficial del php (www.php.net) en D:\php y añadir ';D:\php;' a la variable de entorno PATH, y editar el fichero php.ini que hay en D:\php y darle el siguiente valor a *doc_root*:

doc_root = d:\inetpub\wwwroot

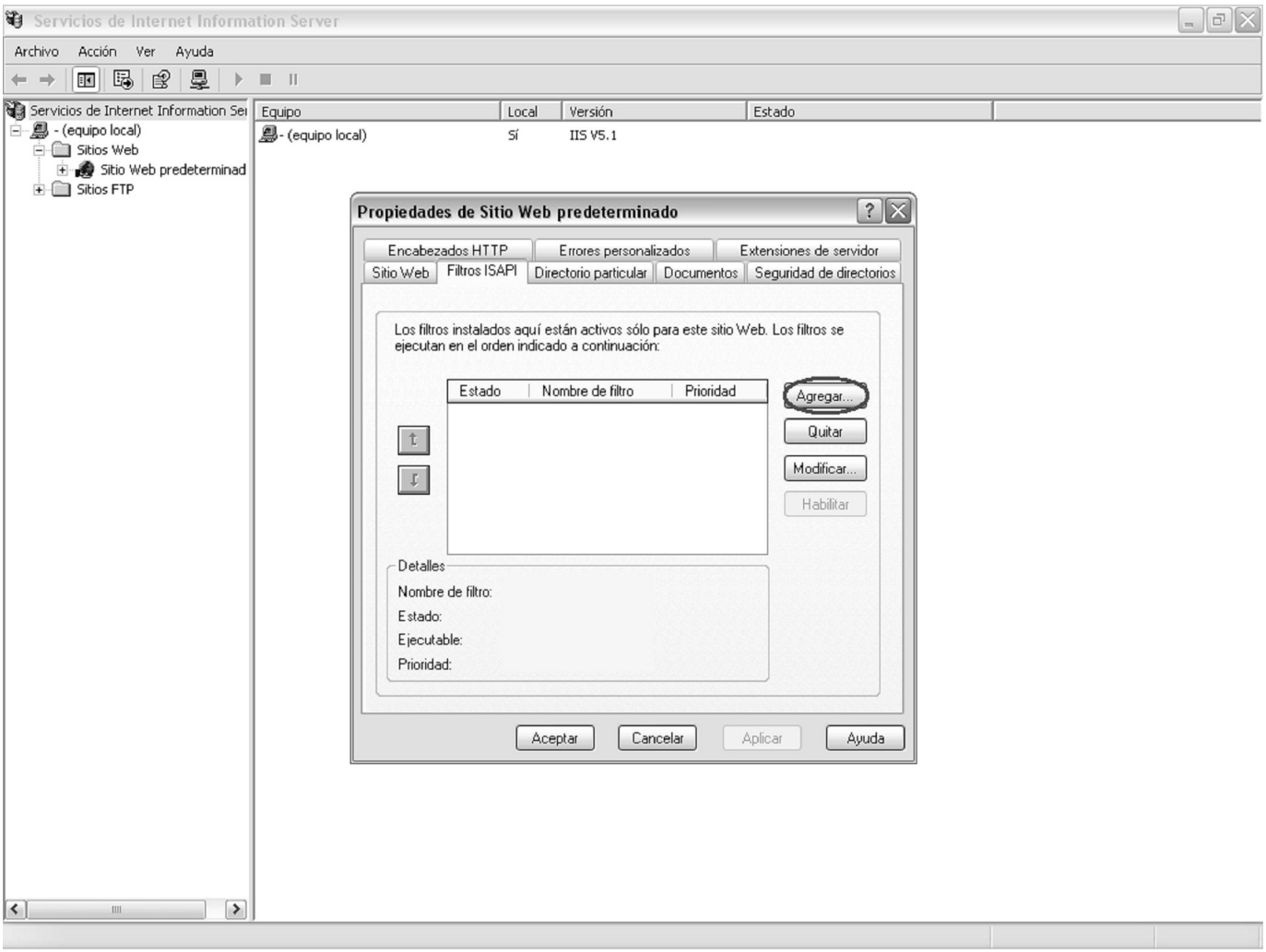
d:\inetpub\wwwroot es el directorio de publicación, en el que estarán los scripts para generar la página web.

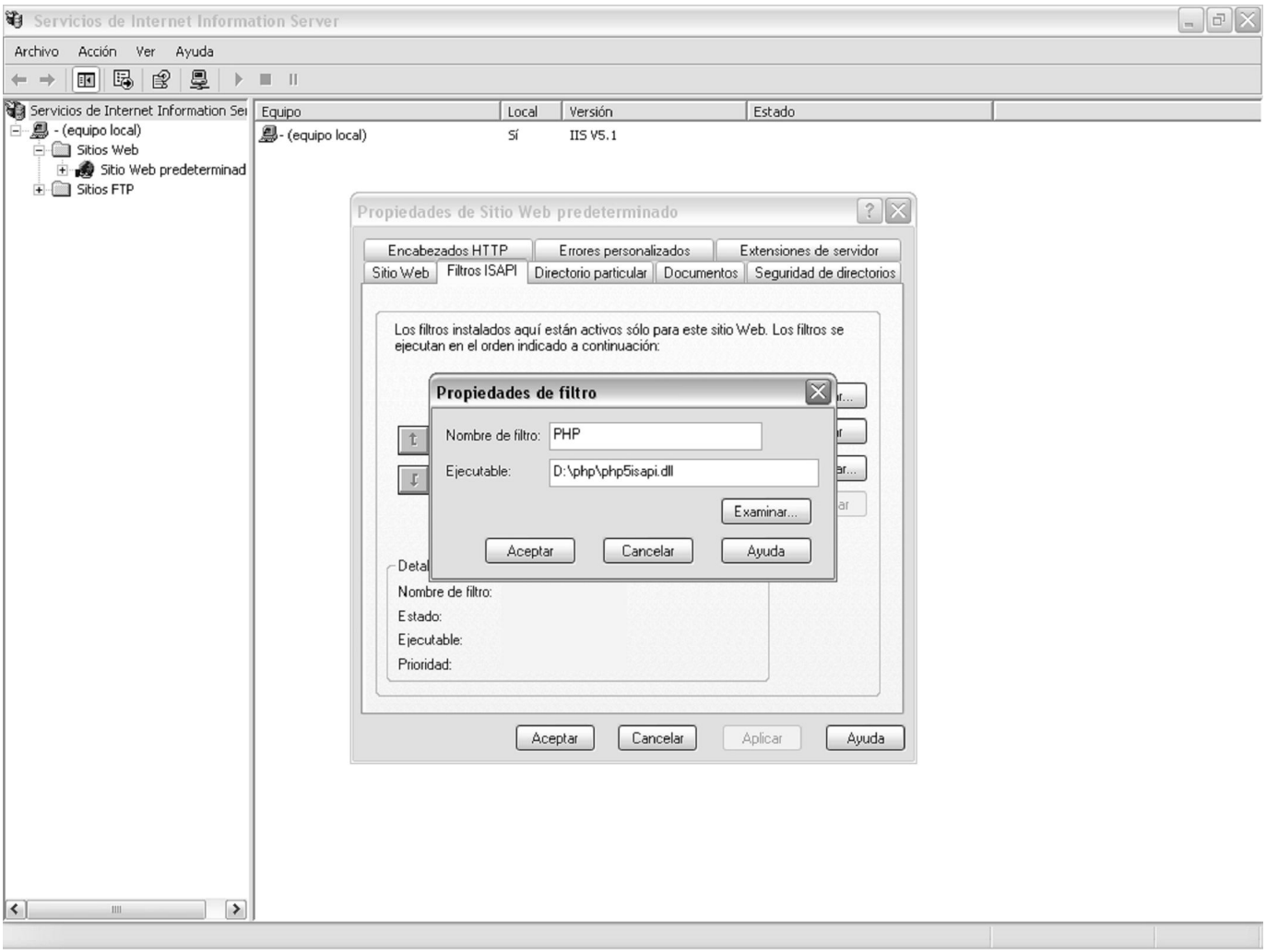
También es necesario activar, en el fichero php.ini, la extensión (dll) que permite usar sockets:

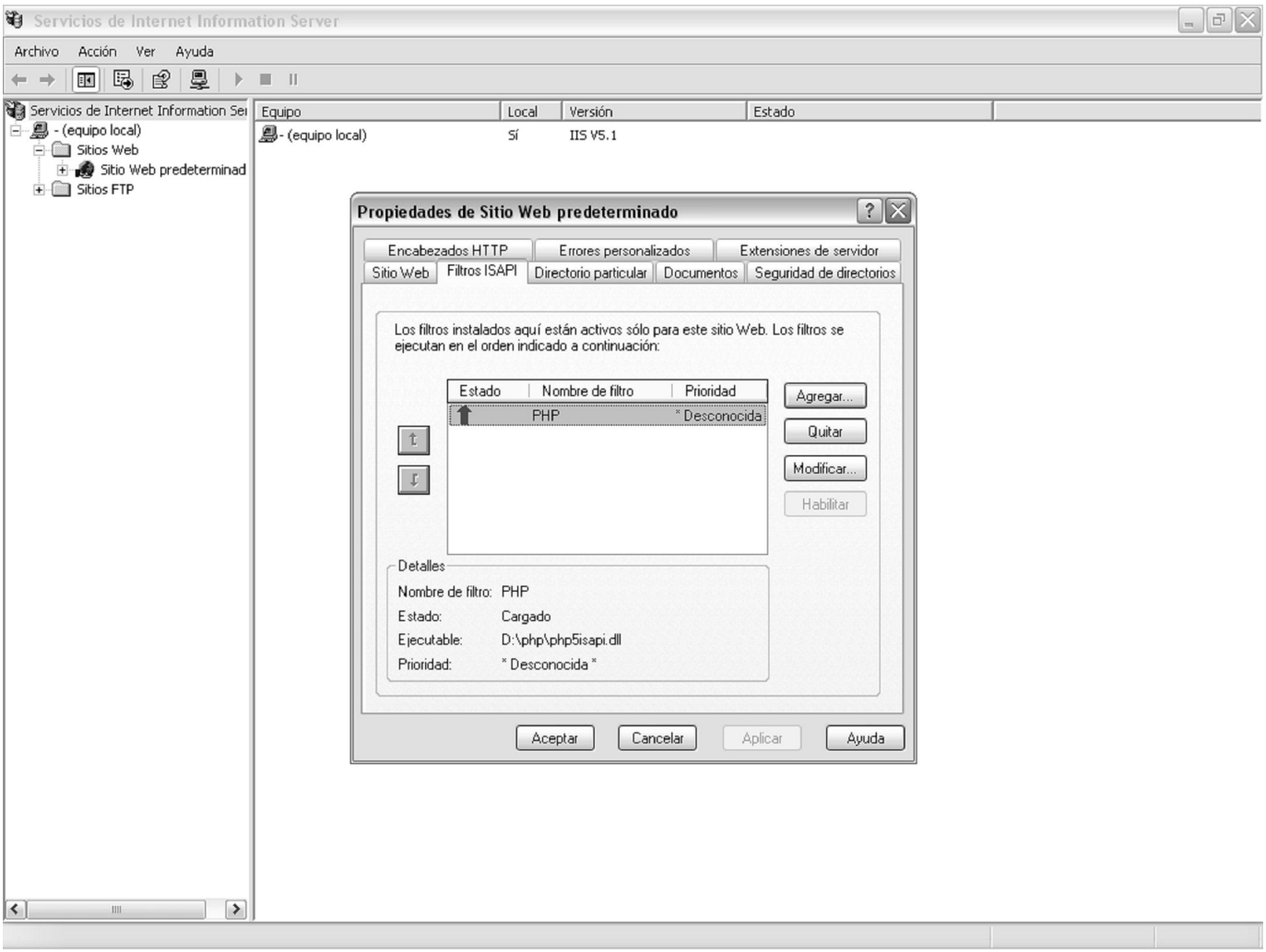
extension = php_sockets.dll

2.1.3. Configuración de IIS

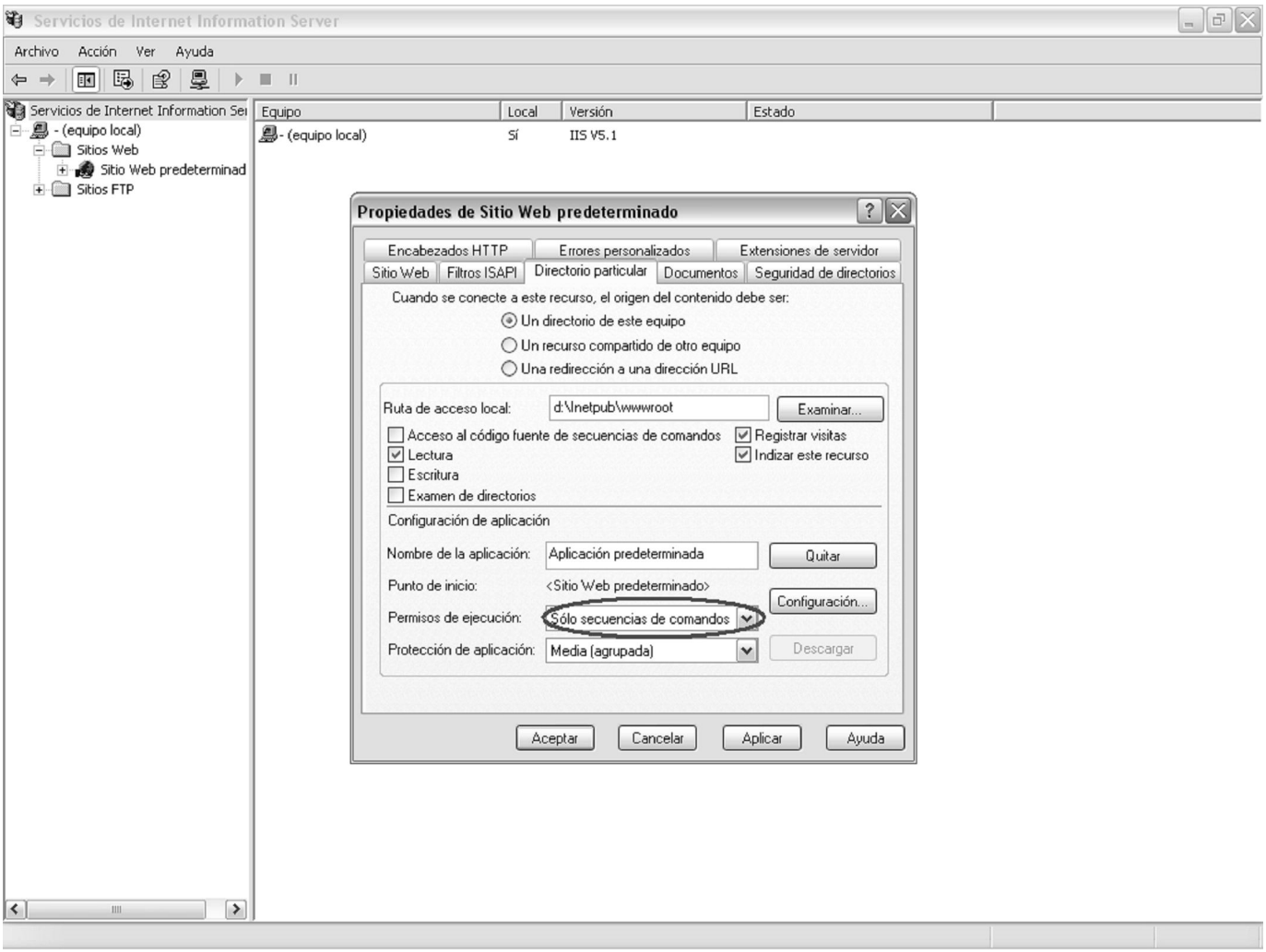
En 'Sitio Web predeterminado', clic con el botón derecho y clic en propiedades. En el cuadro de diálogo, se selecciona la pestaña 'Filtros ISAPI':

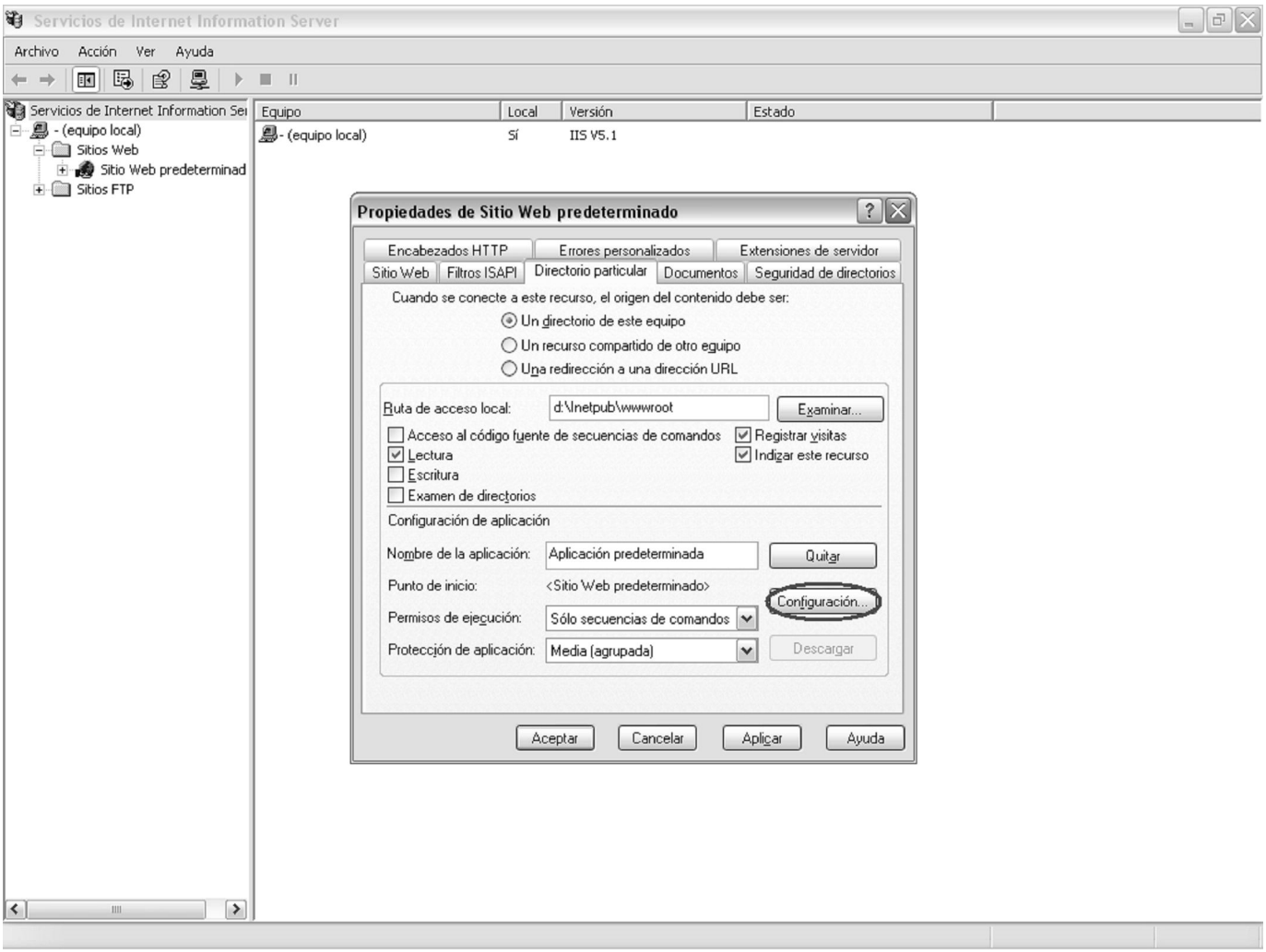


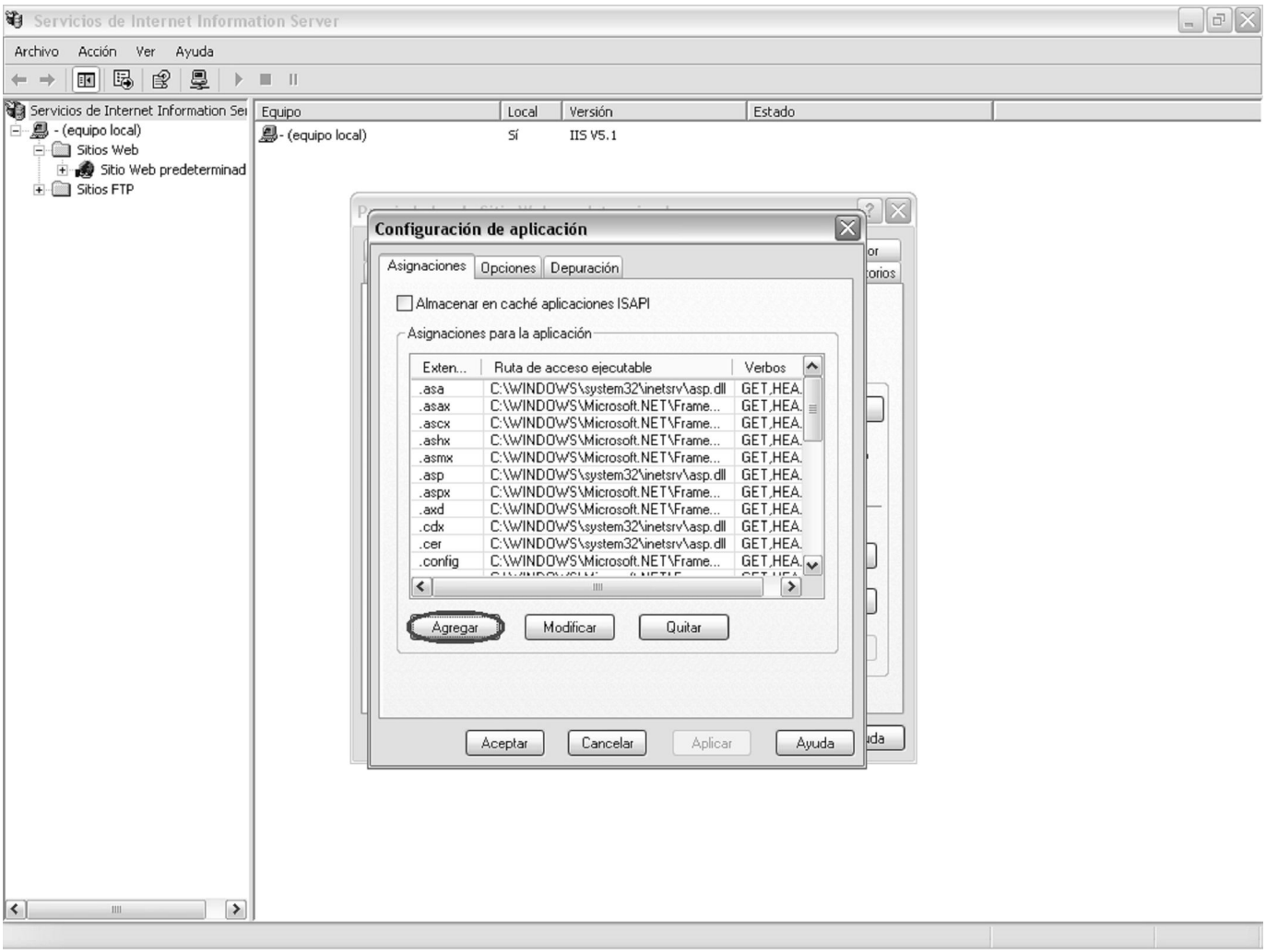

Servicios de Internet Information Server
Archivo Acción Ver Ayuda
Servicios de Internet Information Ser
- (equipo local)
Sitios Web
Sitio Web predeterminad
Sitios FTP
Equipo
Local
Versión
Estado
- (equipo local)
Sí
IIS V5.1
Propiedades de Sitio Web predeterminado
Encabezados HTTP
Errores personalizados
Extensiones de servidor
Sitio Web
Filtros ISAPI
Directorio particular
Documentos
Seguridad de directorios
Los filtros instalados aquí están activos sólo para este sitio Web. Los filtros se ejecutan en el orden indicado a continuación:
Propiedades de filtro
Nombre de filtro:
PHP
Ejecutable:
D:\php\php5isapi.dll
Examinar...
Aceptar
Cancelar
Ayuda
Nombre de filtro:
Estado:
Ejecutable:
Prioridad:
Aceptar
Cancelar
Aplicar
Ayuda

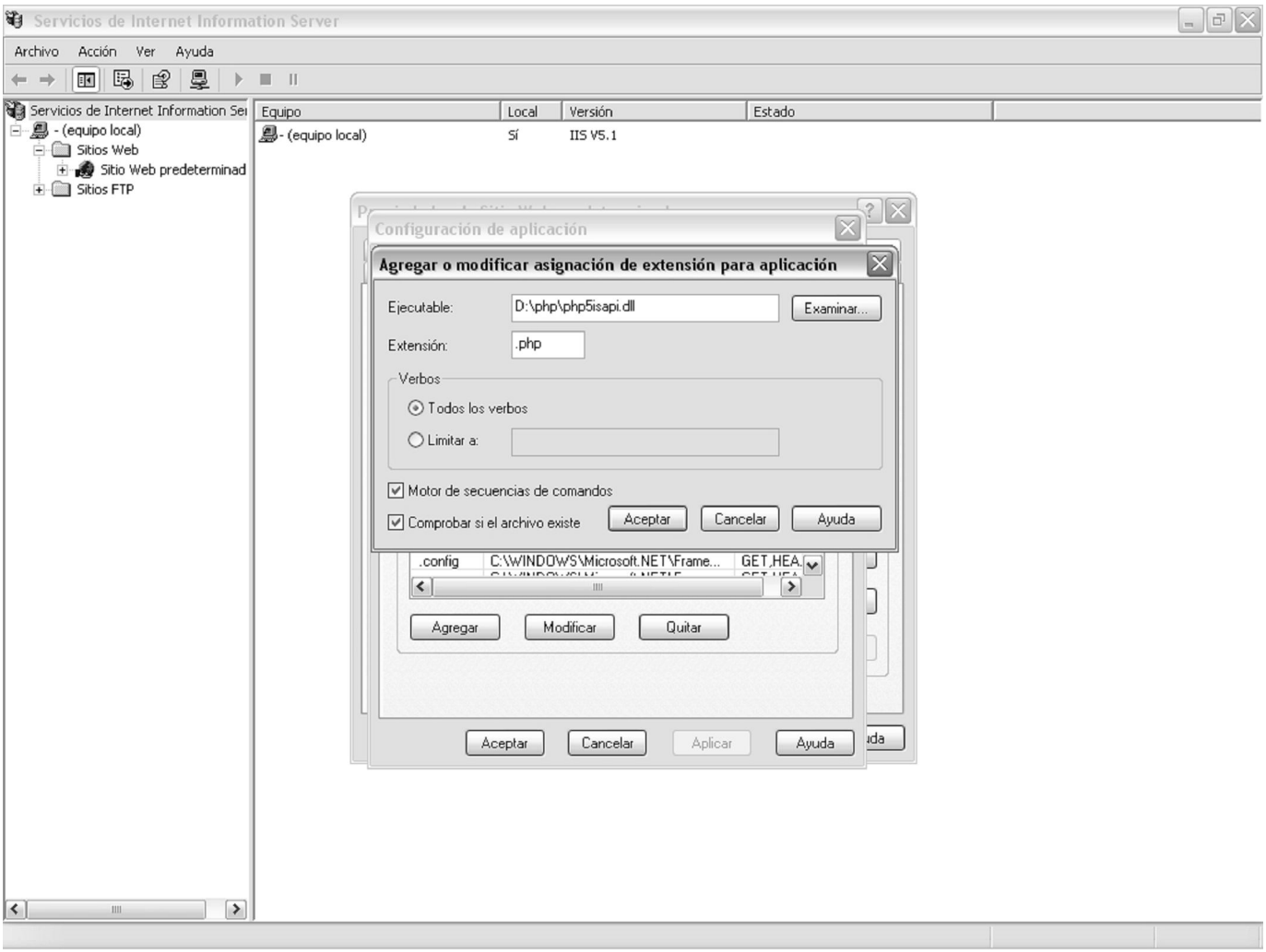


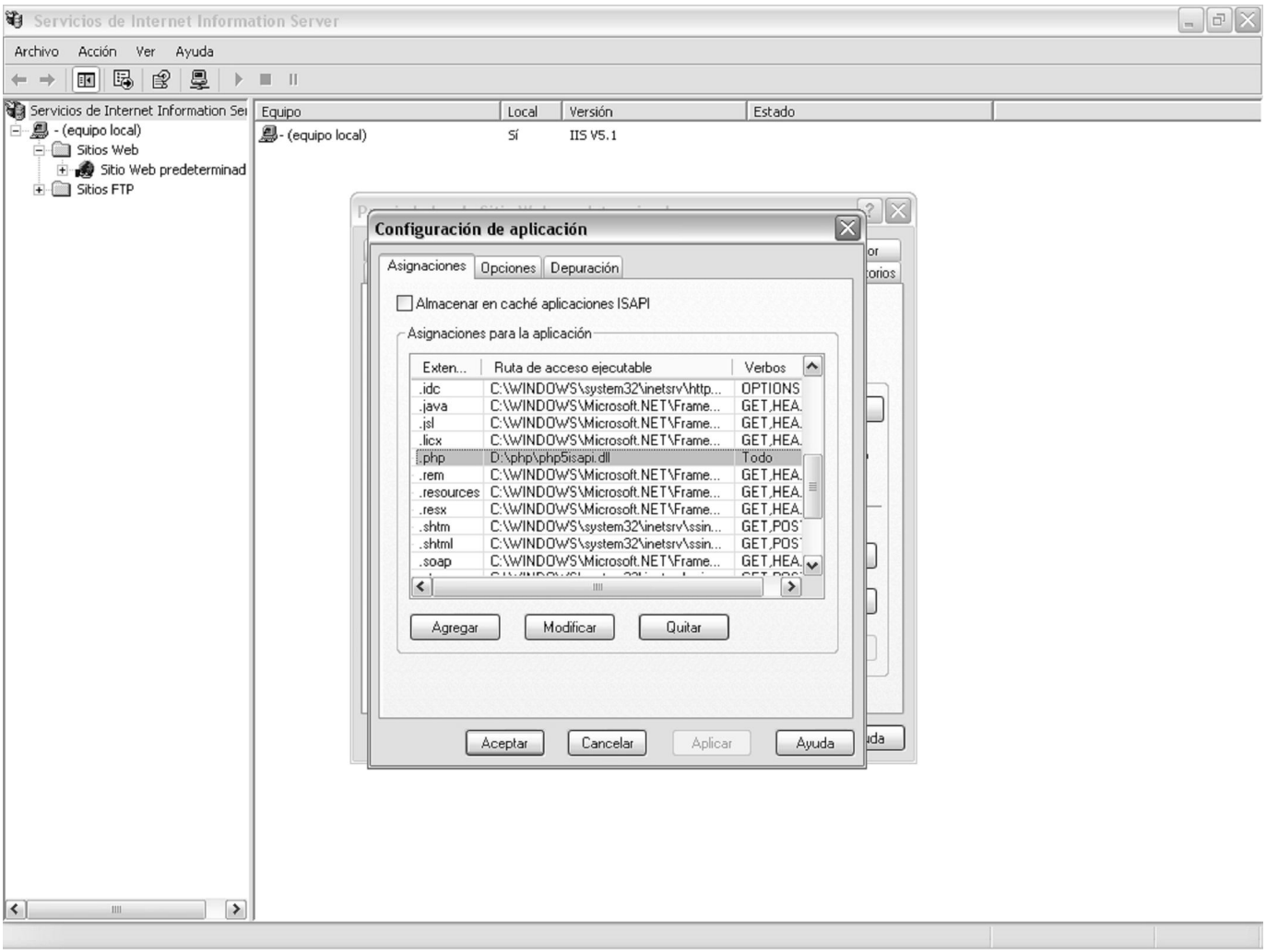
Después, en la pestaña 'Directorio particular':


Servicios de Internet Information Server
Archivo Acción Ver Ayuda
Servicios de Internet Information Ser
- (equipo local)
Sitios Web
Sitio Web predeterminad
Sitios FTP
Equipo Local Versión Estado
- (equipo local) Sí IIS V5.1
Propiedades de Sitio Web predeterminado
Encabezados HTTP
Errores personalizados
Extensiones de servidor
Sitio Web
Filtros ISAPI
Directorio particular
Documentos
Seguridad de directorios
Cuando se conecte a este recurso, el origen del contenido debe ser:
Un directorio de este equipo
Un recurso compartido de otro equipo
Una redirección a una dirección URL
Ruta de acceso local:
d:\Inetpub\wwwroot
Examinar...
Acceso al código fuente de secuencias de comandos
Registrar visitas
Lectura
Indizar este recurso
Escritura
Examen de directorios
Configuración de aplicación
Nombre de la aplicación:
Aplicación predeterminada
Quitar
Punto de inicio:
<Sitio Web predeterminado>
Configuración...
Permisos de ejecución:
Sólo secuencias de comandos
Protección de aplicación:
Media (agrupada)
Descargar
Aceptar
Cancelar
Aplicar
Ayuda

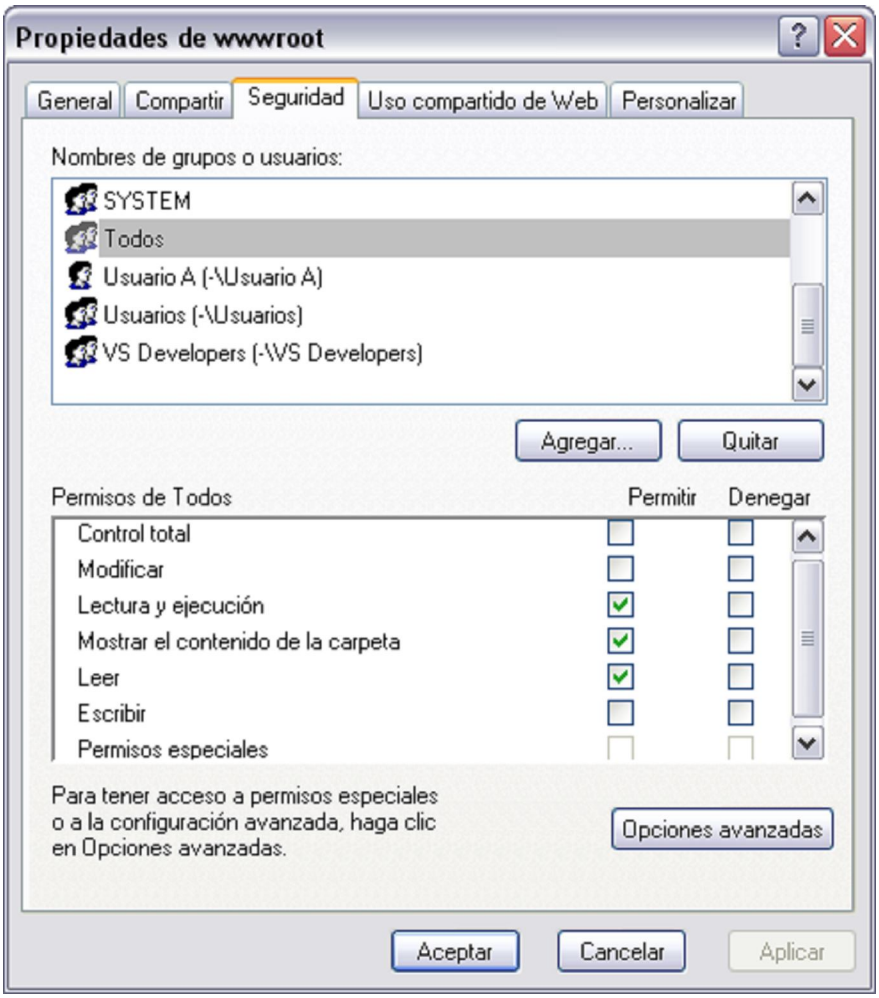

Servicios de Internet Information Server
Archivo Acción Ver Ayuda
Servicios de Internet Information Ser
- (equipo local)
Sitios Web
Sitio Web predeterminad
Sitios FTP
Equipo
Local
Versión
Estado
- (equipo local)
Sí
IIS V5.1
Propiedades de Sitio Web predeterminado
Encabezados HTTP
Errores personalizados
Extensiones de servidor
Sitio Web
Filtros ISAPI
Directorio particular
Documentos
Seguridad de directorios
Cuando se conecte a este recurso, el origen del contenido debe ser:
Un directorio de este equipo
Un recurso compartido de otro equipo
Una redirección a una dirección URL
Ruta de acceso local:
d:\Inetpub\wwwroot
Examinar...
Acceso al código fuente de secuencias de comandos
Registrar visitas
Lectura
Indizar este recurso
Escritura
Examen de directorios
Configuración de aplicación
Nombre de la aplicación:
Aplicación predeterminada
Quitar
Punto de inicio:
<Sitio Web predeterminado>
Configuración...
Permisos de ejecución:
Sólo secuencias de comandos
Protección de aplicación:
Media (agrupada)
Descargar
Aceptar
Cancelar
Aplicar
Ayuda


Servicios de Internet Information Server
Archivo Acción Ver Ayuda
Servicios de Internet Information Ser
- (equipo local)
Sitios Web
Sitio Web predeterminad
Sitios FTP
Equipo
Local
Versión
Estado
- (equipo local)
Sí
IIS V5.1
Configuración de aplicación
Asignaciones
Opciones
Depuración
Almacenar en caché aplicaciones ISAPI
Asignaciones para la aplicación
Exten...
Ruta de acceso ejecutable
Verbos
.asa C:\WINDOWS\system32\inetsrv\asp.dll GET,HEA
.asax C:\WINDOWS\Microsoft.NET\Frame... GET,HEA
.ascx C:\WINDOWS\Microsoft.NET\Frame... GET,HEA
.ashx C:\WINDOWS\Microsoft.NET\Frame... GET,HEA
.asmx C:\WINDOWS\Microsoft.NET\Frame... GET,HEA
.asp C:\WINDOWS\system32\inetsrv\asp.dll GET,HEA
.aspx C:\WINDOWS\Microsoft.NET\Frame... GET,HEA
.axd C:\WINDOWS\Microsoft.NET\Frame... GET,HEA
.cdx C:\WINDOWS\system32\inetsrv\asp.dll GET,HEA
.cer C:\WINDOWS\system32\inetsrv\asp.dll GET,HEA
.config C:\WINDOWS\Microsoft.NET\Frame... GET,HEA
Agregar
Modificar
Quitar
Aceptar
Cancelar
Aplicar
Ayuda


Servicios de Internet Information Server
Archivo Acción Ver Ayuda
Servicios de Internet Information Ser
- (equipo local)
Sitios Web
Sitio Web predeterminad
Sitios FTP
Equipo
Local
Versión
Estado
- (equipo local)
Sí
IIS V5.1
Configuración de aplicación
Agregar o modificar asignación de extensión para aplicación
Ejecutable:
D:\php\php5isapi.dll
Examinar...
Extensión:
.php
Verbos
Todos los verbos
Limitar a:
Motor de secuencias de comandos
Comprobar si el archivo existe
Aceptar
Cancelar
Ayuda
.config
C:\WINDOWS\Microsoft.NET\Frame...
GET,HEA
Agregar
Modificar
Quitar
Aceptar
Cancelar
Aplicar
Ayuda



También es necesario que los permisos de la carpeta de publicación, la carpeta a la que accederán los clientes de la web, sean los adecuados para que éstos puedan ver la página en su navegador:

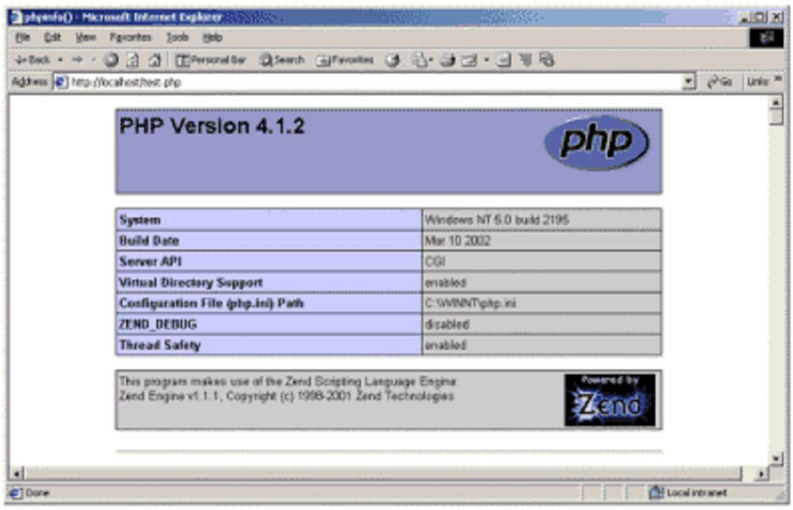


Y finalmente, un test para comprobar que php se ha instalado bien:

Se crea un fichero llamado test.php en la carpeta de publicación (D:\Inetpub\wwwroot) cuyo contenido es:

```
<?
phpinfo();
?>
```

Y se ejecuta un explorador poniendo como dirección http://localhost/test.php. Se ve una pantalla como ésta:



2.2 Elección del servidor de streaming

En Internet se encuentran varias alternativas:

- Darwin Streaming Server: gratis pero sólo disponible para Windows 2000/2003 Server (aparte de para Red Hat Linux y Mac OS X), no para Windows XP. Además, necesita otro programa: un broadcaster que codifique el contenido en directo y se lo entregue al Darwin Streaming Server.
- Windows Media Services: gratis pero sólo viene con Windows 2003 Server y Windows 2000 Server/Advanced Server.
- RealServer: hay que comprarlo si se quiere toda su funcionalidad, o bien está limitado a un número reducido de conexiones de clientes si se quiere usar gratis.
- Windows Media Encoder: gratis y no necesita de ningún otro programa (broadcaster), aunque es recomendable usarlo con Windows Media Services.
- VLC: gratis, no necesita de ningún otro programa (broadcaster) y además es de fácil instalación.

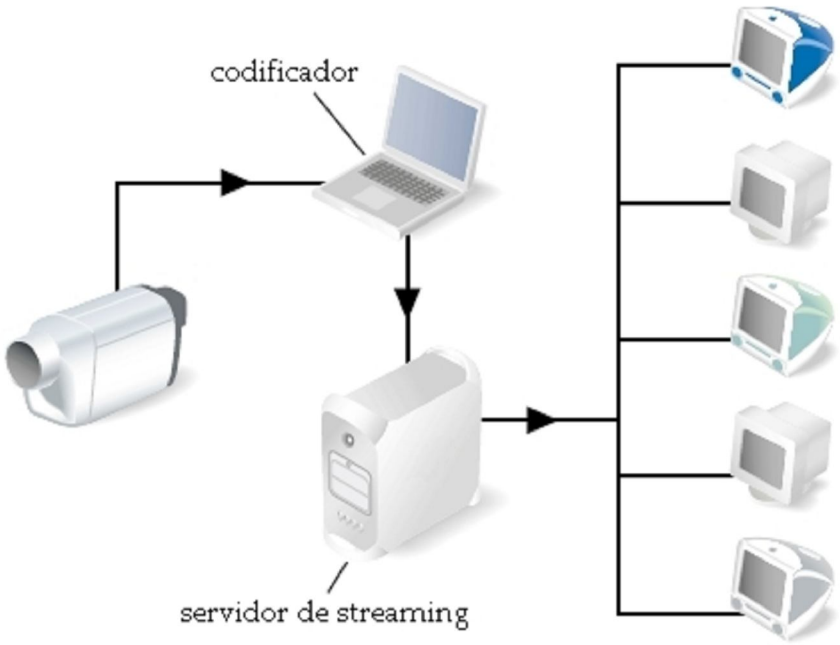
Tanto Windows Media Encoder como VLC son fáciles de utilizar, pero en VLC es necesario recurrir a la línea de comandos si se quiere tener acceso a algunas opciones avanzadas a las que no se puede acceder mediante la interfaz gráfica. Sin embargo VLC tiene muchas más opciones a la hora de codificar, ya que permite elegir el códec, mientras que con Windows Media Encoder no se puede

elegir, ya que utiliza algún códec específico para Windows Media Player. Además, VLC tiene opciones para el encapsulado del stream y para elegir el protocolo mediante el cual va a ser entregado el stream, y puede usarse tanto como servidor de streaming como de reproductor cliente (como el Windows Media Player). Por todo ello, el programa escogido como servidor de streaming fue VLC.

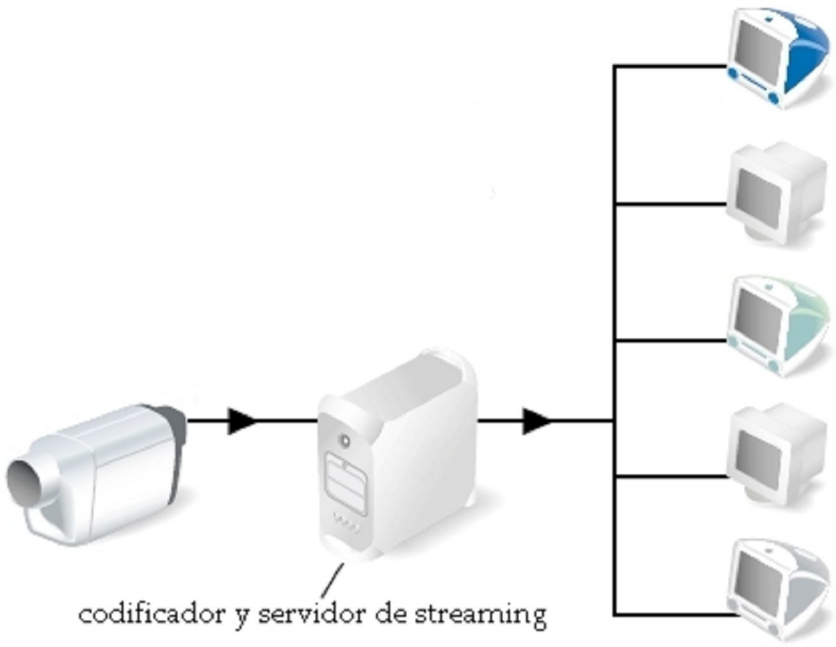
En el servidor hay 3 partes esenciales:

1. Servidor web, gracias al cual el usuario puede ver los Aibos y controlarlos, y también ver lo que están viendo las cámaras de los Aibos.
2. Servidor de streaming que sirve el stream al usuario para que éste vea los Aibos (VLC).
3. Programa que hace de puente entre los clientes y los Aibos, para que los primeros puedan controlar a los segundos (Takepicture).

2.3 Posibles arquitecturas para servir streams



En esta arquitectura, un ordenador con el software de codificación codifica lo que capta la cámara y lo envía al servidor del streaming, que es al que los clientes se conectarán. En el caso de utilizar QuickTime Broadcaster como software de codificación y QuickTime Streaming Server (o su versión de código abierto, Darwin Streaming Server), si se espera que se conecten muchos clientes concurrentemente (por ejemplo a partir de 100), ésta es la arquitectura recomendada. Sin embargo, este software no se puede utilizar en Windows XP (que es el sistema operativo de los ordenadores del laboratorio), por lo que el QuickTime Broadcaster y el Darwin Streaming Server quedan descartados.



Otra arquitectura posible es poner en un mismo ordenador el software de codificación y el software servidor de streaming.

El programa elegido fue VLC por su facilidad de uso y porque no tiene problemas para ser instalado en Windows XP. Este programa incluye tanto el codificador del stream como el servidor de streaming, por lo que la arquitectura empleada se corresponde con la segunda figura.

Además, hemos de montar el servidor web, que puede estar en la misma máquina que sirve el stream o bien en otra máquina. En nuestro caso, estará en la misma máquina.

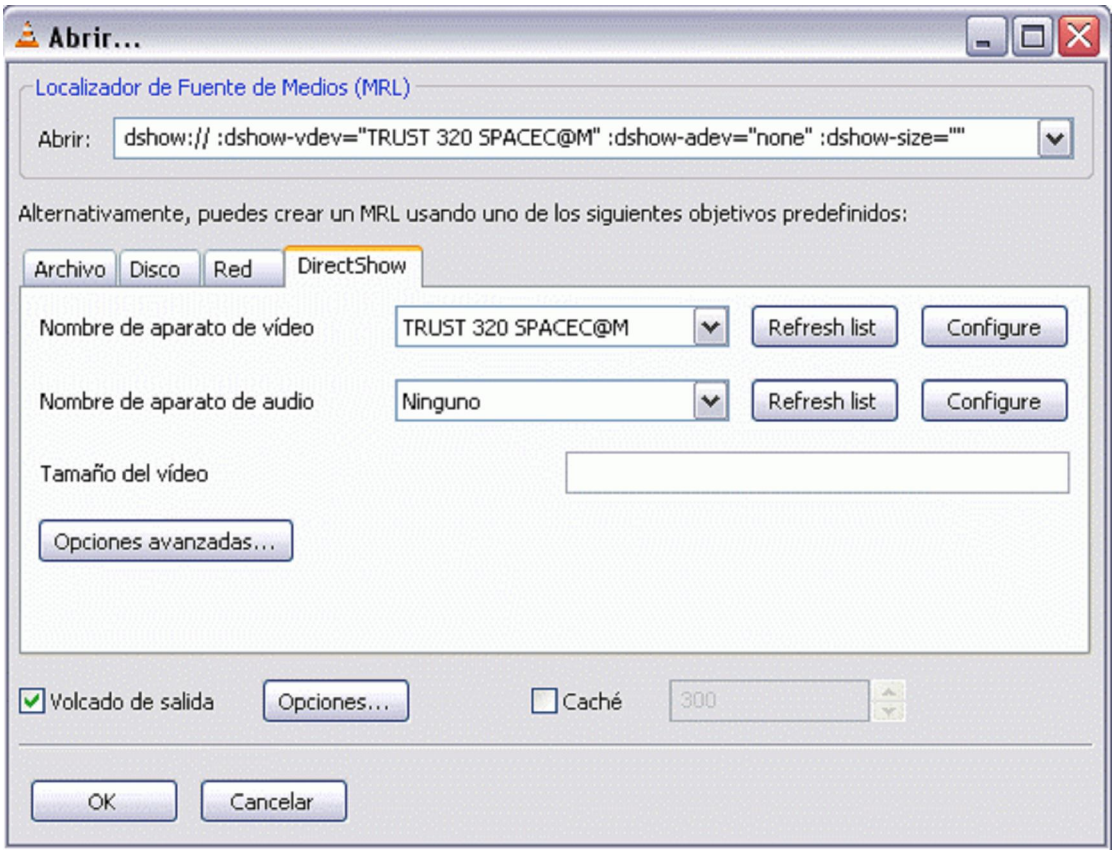
Si se quiere utilizar la misma máquina para que haga tanto de servidor de streaming como de servidor web, se pueden usar 2 interficies de red distintas (ya sea con una misma tarjeta de red para las dos interficies o bien con dos tarjetas de red, una para cada interficie), una para el software servidor web y otra para el software servidor de streaming, con lo que se usaría una IP para el servidor web y otra para el servidor de streaming.

Otra opción es usar la misma interficie tanto para el software de servidor web como para el de servidor de streaming, con lo que usarían los dos la misma IP pero diferente puerto cada uno. Ya que los clientes que se conectan a un servidor web esperan siempre que la conexión se establezca por el puerto

80, habría que configurar el servidor web para este puerto (el predeterminado) y el servidor de streaming para otro puerto. Si el servidor de streaming usa el protocolo http, lo más corriente es usar el puerto 8080, que es el que usaremos. Si usa el protocolo mms (para Windows Media Player), usaremos el puerto 1234.

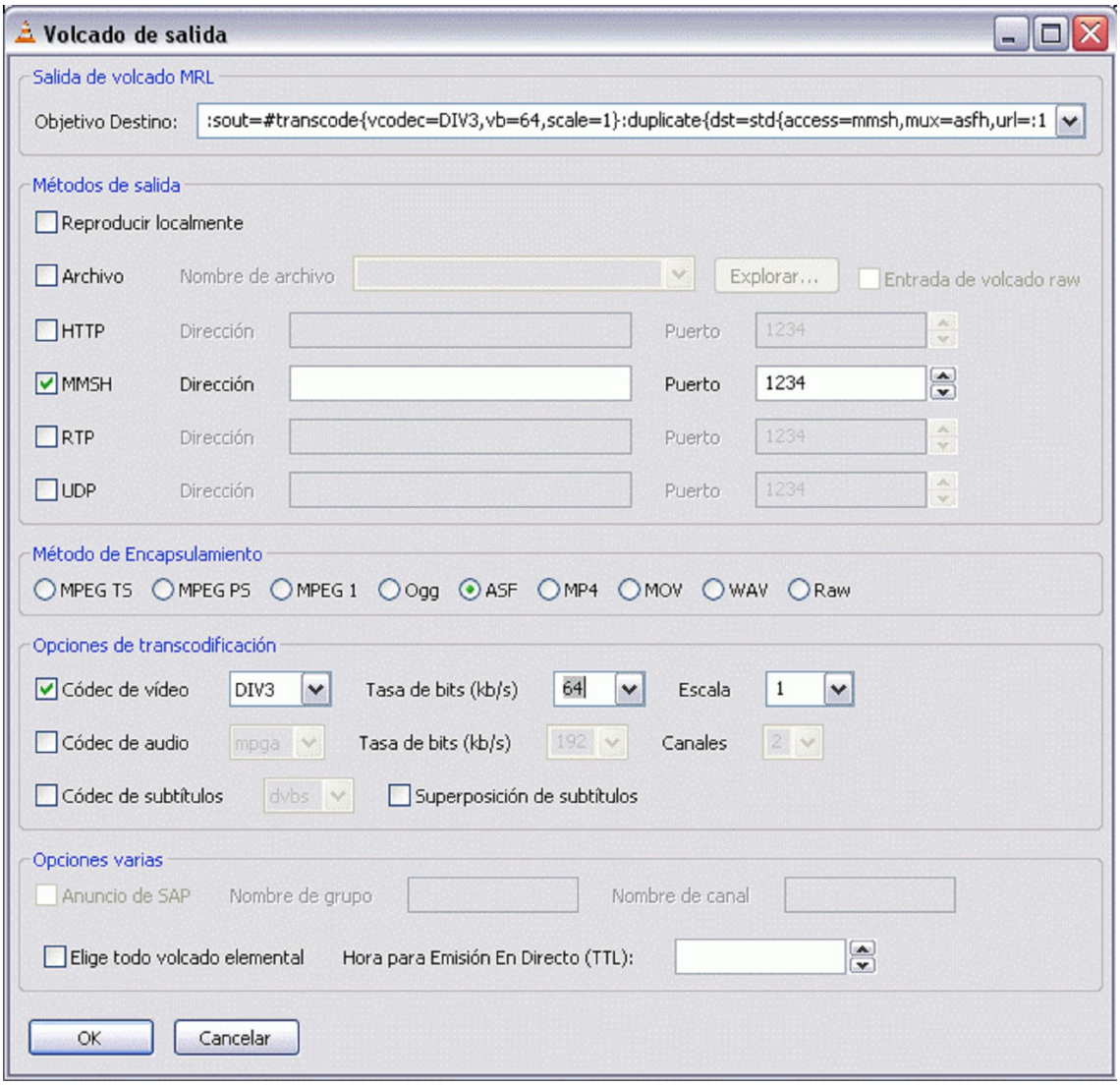
La forma de servir el stream en directo con VLC es la siguiente:

Una vez se hace clic en 'Abrir Aparato de Captura', aparece el siguiente cuadro de diálogo:



En 'Nombre de aparato de vídeo', se elige la webcam que se vaya a usar para captar el contenido en directo, y si se desea audio, se elige también el dispositivo que lo capte en 'Nombre de aparato de audio'.

Se marca entonces la casilla 'Volcado de salida', con lo que se activará el botón 'Opciones'. Pulsándolo, aparece lo siguiente:



Marcamos MMSH y ASF para poder servir a los reproductores Windows Media Player, y seleccionamos un códec (que ha de tenerlo también el cliente) y la tasa de bits. También se puede marcar HTTP para que la entrega del stream se efectúe mediante este protocolo.

2.4. Formas de añadir contenido multimedia a una página web

Hay dos formas de mostrar el contenido multimedia en una página web:

1. Se puede mostrar ejecutándose un reproductor multimedia aparte, fuera del navegador, al hacer clic en algún enlace de la página web, o bien
2. Se puede mostrar incrustado (embebido) en el navegador.

Es interesante para el usuario disponer de las dos opciones, para que pueda elegir lo que prefiera, así que en la página web tiene la opción de escoger.

En este proyecto se ha optado por hacer que el servidor de streaming sirva streams que pueda reproducir el reproductor de Windows Media, por lo que será necesario que los clientes que se conecten tengan instalado este reproductor u otro que sea capaz de reproducir en el formato de windows media video.

Implementación de la **opción 1**: mostrar la secuencia de vídeo en un reproductor **fuera** del navegador

Se crean metaarchivos, que son ficheros de texto con la extensión .wax, .wvx o .asx. Estos ficheros se llaman ficheros redirectores de secuencia, y por lo general, se crea uno por cada fichero del cual se quiere hacer streaming. En nuestro caso no hay ficheros de los que hacer streaming, ya que es contenido en directo. Los .wax son para ficheros del tipo wma (Windows Media Audio), que son ficheros que contienen sólo audio; los .wvx para ficheros wmv (Windows Media Video), que contienen tanto audio como vídeo, y los .asx para fichero asf, que contienen también audio y vídeo creados con versiones anteriores de la tecnología de Windows Media.

Cuando se inserta un enlace a un vídeo en una página web, ha de enlazar a uno de estos metaarchivos, en lugar de enlazar a una URL (Uniform Resource Locator).
Cuando el usuario hace clic en el enlace, el explorador descarga el metaarchivo, para que el contenido de este metaarchivo sea enviado al Windows Media Player, que se conecta al servidor y reproduce el stream.

Estos metaarchivos son necesarios porque no se puede asegurar el comportamiento del navegador si se pone directamente la URL del stream en el enlace, ya que algunos navegadores pueden intentar descargar el contenido como fichero, abrirlo como una página web o simplemente dar un mensaje de error.

```
<ASX version = "3.0">
<Entry>
    <Ref href = "mms://NombreDelServidor/NombreFichero.wmv" />
</Entry>
</ASX>
```

Ejemplo de metaarchivo

Los metaarchivos usan un lenguaje de scripting basado en XML que tiene muchas propiedades y funciones.
En el ejemplo, el metaarchivo instruye a Windows Media Player para que reproduzca el fichero cuyo nombre es *NombreFichero.wmv*:

El tag ASX define a lo que tiene incluido dentro como parte de un metaarchivo de Windows Media.

El tag Ref especifica, mediante el atributo href, la URL donde encontrar el stream, y en este caso se usa el protocolo de streaming MMS.

El metaarchivo puede contener múltiples tags Entry, permitiendo crear así listas de reproducción. Además puede contener también tags que controlan la manera en que se reproduce el contenido digital y otros que añaden propiedades de texto y links a la interfaz del reproductor de Windows Media.

Una vez creados los metaarchivos, hay que publicarlos en el servidor web, es decir, ponerlos en un directorio que sea accesible desde la página web, y añadir los enlaces a ellos en la página web, que como el del siguiente ejemplo:

```
Haga clic <a href="http://NombreDelServidorWeb/NombreFichero.wvx"> aquí </a> para
reproducir la secuencia.
```

Ejemplo de enlace a un metaarchivo

La ventaja de que se abra un reproductor aparte del navegador es que el cliente puede seguir navegando a otros sitios mientras el reproductor reproduce el stream, además de disponer de todas las características del reproductor. Para el creador de la web, la ventaja es la simplicidad del código.

Implementación de la **opción 2**: mostrar la secuencia de vídeo **dentro** del navegador

En este caso sí es necesario centrarse en un reproductor concreto, no sirve sólo que sea capaz de reproducir el contenido, también ha de ser un reproductor que se pueda embeber en el navegador. Se ha elegido el Windows Media Player, por lo que en este caso, el usuario cliente sí que necesita tener el Windows Media Player instalado.

Con esta alternativa, la principal ventaja del creador de la página web es que tiene completo control sobre el aspecto del reproductor y sobre sus funciones, por ejemplo, poniendo de manifiesto sólo aquellas funciones que sean apropiadas para la página. Por ejemplo, en este proyecto bastaría con mostrar los botones de *play* y de *stop*, no sería necesario mostrar los de *avance* y *rebobinar*, ya que sólo se mostrará vídeo en directo.

Otra ventaja es que se pueden añadir comandos que permiten controlar también el resto de la página web con el reproductor. Por ejemplo, se pueden añadir comandos al stream que cambien imágenes en un frame o que envíe comandos DHTML (HTML dinámico) al navegador de forma sincronizada con el stream.

Lo primero es añadir un tag OBJECT para el control ActiveX del Windows Media Player. El tag OBJECT hará que se abra el control ActiveX del reproductor en la página web cuando los clientes que tengan instalado el Windows Media Player entren en la página web.

Ejemplo:

```
<HTML>
<HEAD>
<TITLE>Control de Windows Media Player embebido en el navegador</TITLE>
</HEAD>
<BODY>
<OBJECT ID="Player" width="320" height="240"
  CLASSID="CLSID:6BF52A52-394A-11d3-B153-00C04F79FAA6">
</OBJECT>
</BODY>
</HTML>
```

En el ejemplo, los atributos width y height del tag OBJECT determinan, respectivamente, la anchura y la altura de la imagen.
Es importante que el tag OBJECT se encuentre dentro de la sección BODY de la página web, entre los tags <BODY> y </BODY>. En caso de querer esconder los controles de la interfaz de usuario, hay que poner los valores de los atributos width y height a 0, o bien asignar el valor “invisible” a la propiedad *Player*.uiMode mediante los tags PARAM que se comentarán más adelante, ya que colocar el OBJECT en la sección HEAD de la página para esconder la interfaz de usuario puede provocar resultados inesperados.

El valor del atributo CLASSID indica, en este caso, que la versión del Windows Media Player ha de ser, al menos, la 7, por lo que si un usuario tiene instalada una versión más antigua, el reproductor no se abrirá. Se puede hacer que si el cliente no tiene al menos la versión 7 instalada, se abra una ventana con un mensaje, con un script en VBScript como el siguiente:

```
<SCRIPT LANGUAGE="VBScript">
<!--
 On error resume next
   Player.URL = ""
   if err then msgbox "Necesita tener instalado Windows Media Player 7." & chr(13)
&_
        "Diríjase a http://www.microsoft.com/windowsmedia"
```

```
     err.clear
-->
</SCRIPT>
```

Si el Windows Media Player 7 no se puede iniciar, se lanza un error cuando se llama al método Player.URL, y la sentencia If comprueba la presencia de un error y abre la caja de texto en caso afirmativo.

También se puede, en lugar de usar el control ActiveX para la versión 7 de Windows Media Player, usar el control para la versión 6.4, ya que cualquier script compatible con la versión 6.4 lo será también con la 7. Además hay que tener en cuenta que no todos los usuarios pueden instalar Windows Media Player 7, por ejemplo los usuarios de Windows 95 o de Windows NT, o que no existe el plugin de la versión 7 para algunos exploradores web que utilizan algunos usuarios, por ejemplo para el Netscape.

El código sería como éste:

```
<OBJECT ID="MediaPlayer" WIDTH=320 HEIGHT=240
  CLASSID="CLSID:22D6f312-B0F6-11D0-94AB-0080C74C7E95"
  STANDBY="Cargando componentes de Windows Media Player"
  TYPE="application/x-oleobject">

<EMBED TYPE="application/x-mplayer2"
    NAME="MediaPlayer"
    WIDTH=320
    HEIGHT=240>
  </EMBED>

</OBJECT>
```

En este caso, el valor del atributo CLASSID de OBJECT es el de la versión 6.4 del Windows Media Player, y hay un tag EMBED que sirve para los usuarios de Netscape.

Además se puede añadir el atributo CODEBASE a OBJECT para que se verifique en el ordenador del usuario si está instalado Windows Media Player, y si no lo está, o no está instalada la versión que se indique en CODEBASE (en el siguiente ejemplo, la 6.4), se instalará automáticamente desde la dirección indicada en la URL:

```
CODEBASE="http://activex.microsoft.com/activex/controls/mplayer/en/nsmp2inf.cab#Version=6,4,7,1112"
```

Además de usar VLC como servidor de streaming, también puede ser usado por el cliente para ver el vídeo incrustado en el navegador, pero sólo es posible en los navegadores Mozilla y Firefox, instalándoles un plugin, por este motivo se descartó VLC como reproductor incrustado para el navegador del cliente (incrustado en la página web). Sin embargo, poder incrustar VLC en el navegador habría sido lo ideal porque las pruebas hechas para RTP y multicast con VLC como servidor de streaming tuvieron éxito usando como cliente también VLC, sin embargo no tuvieron éxito usando como cliente Windows Media Player.

Los objetos COM

El Component Object Model (COM) es un estándar para la interfaz de objetos. La interfaz es la manera que tiene un programa de comunicarse con un objeto. Por definición, un objeto COM sólo utiliza **propiedades** y **métodos** como interfaz. Algunas propiedades son de sólo lectura, otras de lectura/escritura.

Con ActiveX se puede programar páginas dinámicas, tanto del lado del servidor como del cliente, aunque con diferencias en los dos casos:
Cliente: son programas incluidos en la página web, parecidos a los applets de Java en su funcionamiento, aunque un Applet de Java no puede tomar privilegios para ejecutar acciones peligrosas (o potencialmente peligrosas) en el ordenador donde se ejecuta, y a un control ActiveX sí

que se le pueden otorgar permisos para que sea capaz de hacer cualquier cosa. Esto puede ser un riesgo para la seguridad en el ordenador del cliente.

Los controles ActiveX se usan en este proyecto para invocar al reproductor Windows Media Player. Existen plugins para navegadores que no son Internet Explorer, como Mozilla, Firefox y Opera, que añaden soporte para los controles ActiveX, de manera que también se puede incrustar el reproductor Windows Media Player en estos navegadores.

Para Linux se han hecho también controles ActiveX: los Reaktivate, para el navegador Konqueror, pero los ActiveX de Windows Media Player no son compatibles con los Reaktivate.

2.5. Comunicación entre el servidor web y la aplicación que controla al Aibo

2.5.1. Una forma rudimentaria de comunicación

En un principio, se pensó en una forma rudimentaria la comunicación entre el servidor web con php y la aplicación que controla al Aibo, que gracias al Remote Framework puede ser una aplicación en C++. Dado que la aplicación C++ iba a ejecutarse en la misma máquina que hace de servidor php, se podrían usar ficheros para que una aplicación se comunique con la otra. Si, por ejemplo, el servidor web debe darle alguna instrucción a la aplicación C++, se puede hacer que la aplicación de php escriba esa instrucción en algún fichero que la aplicación C++ pueda leer más tarde. La aplicación C++ debería ir leyendo el fichero cada cierto tiempo por si se ha modificado para dar alguna instrucción nueva. Sería necesario que el fichero contuviera, además de la instrucción, un *timestamp* para que la aplicación C++ ejecutase la instrucción que le toca ejecutar en cada momento, sin repetir instrucciones que ya se le han dado y sin dejar de hacer caso a instrucciones nuevas.

Además, para que la aplicación C++ pudiese recibir todas las instrucciones, sería más apropiado que se fuesen añadiendo las instrucciones nuevas al fichero, con lo que éste tendría una lista de instrucciones que la aplicación C++ debería procesar. Si no se hiciese así, sino que se escribiese la nueva instrucción en el fichero borrando la antigua cada vez (por ejemplo creando cada vez un nuevo fichero que contuviese una sola instrucción), sería posible que antes de que la aplicación C++ leyera una instrucción del fichero, ésta fuese eliminada al crearse el nuevo fichero con otra nueva instrucción, con lo que la aplicación C++ se saltaría instrucciones y el Aibo dejaría de recibirlas.

La aplicación C++, al leer el fichero, debería descartar de alguna manera las instrucciones que ya hubiera leído de él, para lo que podría recordar el *timestamp* de la última instrucción ejecutada.

Dado el siguiente ejemplo de fichero, en el que las instrucciones se guardan en orden cronológico empezando por la más antigua:

18:24 Izquierda

18:25 Derecha
18:26 Adelante

La aplicación C++ debería leer las instrucciones empezando por el final (primero la de las 18:26), y acabando de leer en cuanto detectase una instrucción que ya hubiera leído. Leyéndolas así, evitaría volver a leer las instrucciones que ya había leído antes. La aplicación guardaría la hora 18:26 y, además, el número de veces que ha leído una hora igual a la última, en este caso, las 18:26, por lo que guardaría un 1, ya que se la última hora (las 18:26) la lee sólo una vez (un poco más adelante veremos para qué sirve esto); la próxima vez que leyese el fichero pararía de leer en la línea del fichero que tuviera una hora menor que esa.

Si ahora la aplicación php escribe 2 comandos más en el fichero:

18:24 Izquierda
18:25 Derecha
18:26 Adelante
18:26 Derecha
18:26 Adelante

La aplicación C++ debe leer: 18:26 Adelante, 18:26 Derecha, 18:26 Adelante, 18:25 Derecha, y además, debe detectar que estas dos últimas ya las había leído antes. Como antes había guardado un 1 (el número de veces que había leído las 18:26), ya sabe que a partir de la línea en que pone '18:25 Derecha' debe descartar **una** línea (por el 1 guardado) en la que pone las 18:26.

Esta manera de resolver el problema de comunicación entre las dos aplicaciones fue descartada, ya que se encontró una solución mejor en una de las aplicaciones que trae consigo el Remote Framework: la aplicación Takepicture.

2.5.2. Programa para controlar Aibos: Takepicture

El **AIBO Remote Framework** es un entorno de aplicación en Windows que permite crear software para Windows que permita controlar un AIBO (ERS-7) de forma remota mediante una LAN inalámbrica.

En el Remote Framework se encuentran algunos programas en C++ hechos para controlar Aibos. Con uno de ellos, llamado **Takepicture**, se puede controlar **un** Aibo y hacer que tome fotos y las envíe al ordenador que lo controla. Como este programa está hecho para controlar sólo un Aibo, hubo de ser modificado para poder controlar más de uno. En principio se podía hacer de forma que la interfaz con el usuario, que sólo pedía la IP de un Aibo, fuese cambiada por otra interfaz que pidiese más IPs, pero el problema es que haciéndolo de esta manera, quedaba limitado el número de Aibos que se podían controlar. Así, lo que se ha intentado es dejar la misma interfaz, o muy parecida a la original, ya que se han modificado algunas cosas, por ejemplo, que el usuario no tenga que introducir el nombre de usuario y la clave, y se han eliminado algunos botones en los que era necesario hacer clic para que el proyecto funcionase, haciendo que las funciones que se llamaban al activar estos botones se llamen ahora siempre al iniciarse la aplicación; pero básicamente, la interfaz es casi igual a la del programa original, con lo que una instancia de la aplicación sirve, como en el programa original, para controlar sólo un Aibo; pero ahora, si se ejecutan más instancias de Takepicture, pueden controlarse otros Aibos (uno diferente por cada instancia que se ejecute), lo que no era posible con el programa original.

También es posible controlar un mismo Aibo con dos o más instancias diferentes de Takepicture, si a cada instancia se le indica la IP del mismo Aibo, pero no es algo que necesitemos. Consideraremos que una instancia de Takepicture controla un Aibo y que un Aibo es controlado por una instancia de Takepicture, y no consideraremos el caso en que un Aibo sea controlado por más de una instancia de Takepicture.

Takepicture crea un socket en el servidor a través del cual se comunica con el programa en php. El programa en php crea una instancia de la clase AIBOControl por cada Aibo que se puede controlar por web. En el programa php, al crear una instancia de esta clase (es decir, en el constructor), se indica el puerto que la instancia de Takepicture estará usando para recibir las instrucciones o comandos que se quiere que esa instancia de Takepicture envíe al Aibo que controla; por cada Aibo que se controle, Takepicture usará un puerto diferente en el servidor; es decir, que cada instancia de Takepicture usará un puerto distinto para recibir las órdenes que le envíe el script php. Recogiendo un parámetro

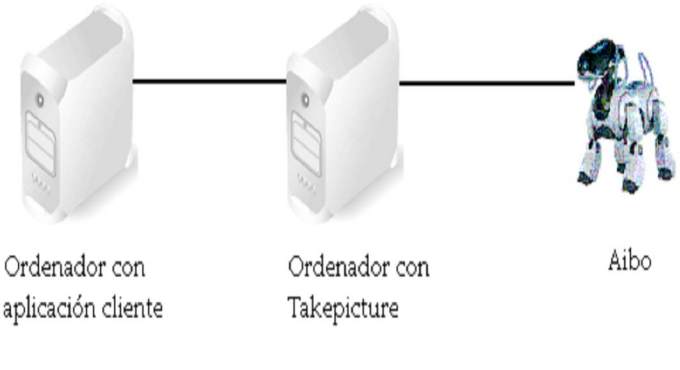
(IDAIBO) que el usuario nos envía mediante el formulario web, el script php identifica cuál es el Aibo al que ha de ir dirigido el comando de control (que también se recoge con el mismo formulario en COMMAND) y en consecuencia crea una instancia de AIBOControl con un puerto u otro para que dirija el envío del comando a ese puerto del ordenador en que se ejecuta Takepicture:

```
// --------------------
// # MANEJO DEL AIBO: #
// --------------------
if($COMMAND !="" || $COMMANDH !="") {
      require_once("RFW_aiboControl.php");

      if(is_numeric($IDAibo)) {
            if($IDAibo >= 1) $aibo = new AIBOControl(54320 + $IDAibo);
            else die ();
      }
      else die ();
```

Como se puede observar, los puertos que las instancias de Takepicture usarán para recibir los comandos son desde el 54321 en adelante.

Takepicture, como se ha explicado con anterioridad, es una aplicación creada en C++ que permite enviar instrucciones a un Aibo utilizando la librería Remote Framework. Con esta aplicación se puede controlar un Aibo de dos maneras: mediante una interfaz de usuario con botones que determinan lo que el Aibo debe hacer, o bien controlando la aplicación Takepicture desde otra aplicación cliente, ya esté ésta ejecutándose en la misma máquina que Takepicture o en una máquina remota.

	
Aplicación cliente en máquina remota	Aplicación cliente en la misma máquina que Takepicture

La comunicación entre la aplicación cliente y Takepicture se realiza mediante **sockets**.

El propósito de los sockets es hacer abstracción de los protocolos de la red subyacente de manera que no sea necesario conocerlos en detalle y que la aplicación pueda funcionar en cualquier red que soporte sockets.
Un socket es un punto final de comunicación, un **objeto** a través del cual una aplicación que usa sockets envía o recibe paquetes de datos a través de una red. Normalmente, los sockets intercambian datos sólo con otros sockets en el mismo ámbito de comunicación.

Las librerías MFC tienen soporte para la programación de los Windows Sockets gracias a dos clases: **CSocket**, que es sencilla de utilizar pero un poco limitada, y **CAsyncSocket**, que es más avanzada y es la que se utiliza en Takepicture. La especificación de los Windows Sockets es un estándar de red abierto que está basado en la implementación de los sockets UNIX.

Hay dos tipos de socket, y los dos son bidireccionales: los datos pueden viajar en ambas direcciones simultáneamente.
Los dos tipos de socket son:

Stream socket: proporciona un stream de bytes que no está limitado a tamaños de registro concretos, que se garantiza que el flujo de bytes será entregado en el mismo orden en el que fueron enviados, sin pérdidas ni duplicados.

La capa de transporte de red puede fragmentar o agrupar datagramas, pero CSocket y CAsyncSocket reagrupará o fragmentará los datagramas en los casos necesarios de forma automática.

Datagram socket: soporta paquetes de datos de tamaños concretos, y no se garantizan ni la entrega de los paquetes ni que el orden de llegada sea el mismo que el de envío.

Los objetos socket de MFC encapsulan un manejador de eventos, cuyo tipo de datos es **SOCKET**. Por ejemplo, cuando se crea un objeto de la clase CAsyncSocket, el objeto encapsula un manejador de eventos Windows SOCKET y proporciona operaciones para ese manejador de eventos.

Takepicture usará las siguientes operaciones de manejo de eventos: OnReceive, OnSend, OnAccept, OnClose.

Estas funciones están declaradas dentro de la clase CAsyncSocket como funciones virtuales, por lo que se podrá definir lo que debe hacer cada una de ellas en los casos en que se produzcan sus eventos asociados.

Ya se ha observado que el orden en que el Aibo debe ejecutar las acciones ha de ser el mismo en el que la aplicación cliente envía a Takepicture los comandos (del mismo modo que en la primera implementación sin sockets debía leerse el fichero en un orden determinado), y que además los **stream sockets** proporcionan mecanismos automáticos para ello, y es por esto por lo que en Takepicture se utilizan este tipo de sockets.

Secuencia de operaciones necesarias para una comunicación mediante un Stream Socket:

A continuación se muestra la secuencia de operaciones que debe seguir un socket en Takepicture, que actúa como servidor (socket servidor):

Servidor	**Cliente**
`// construir socket`	`// construir socket`

CAsyncSocket sockSrvr;	CAsyncSocket sockClient;

// crear el SOCKET
 	
sockSrvr.Create(*númeroDePuerto*);
	// crear el SOCKET
	
	sockClient.Create( );

// comenzar escucha
 	
sockSrvr.Listen( );	

	// buscar conexión sockClient.Connect(*dirección de red del socket a la que se conectará este objeto. Puede ser un nombre de dominio o una IP, puerto que usará esta aplicación cliente*);

// construir un nuevo socket vacío
 	
CAsyncSocket sockRecv;
 	
// aceptar conexión
 	
sockSrvr.Accept(sockRecv);
 	
sockRecv es una referencia al nuevo objeto creado con CSocket, ya que si no se pasara una referencia, en cuanto se saliera del ámbito en que se crea el objeto, éste se destruiría y la conexión se cerraría. La MFC conecta el nuevo objeto a un manejador de eventos de SOCKET.	

Cuando se ha de especificar un puerto en alguna función, es para que se identifique de manera única la aplicación que usa los sockets (asociando el socket a un puerto) y así poder tener varias aplicaciones que usen sockets ejecutándose en una misma máquina al mismo tiempo. Cada aplicación deberá usar unos puertos diferentes. Esto deberá tenerse en cuenta si quiere ejecutarse más de una instancia de Takepicture. En principio, Takepicture sólo usaba un puerto concreto para la comunicación con la aplicación cliente que le da instrucciones para el Aibo, con lo que si se ejecutaban varias instancias de Takepicture en el mismo ordenador, todas intentaban utilizar el mismo puerto (el TCP 54321). Como se comentó anteriormente, esto hubo de ser modificado para que cada una de las instancias utilizase un puerto distinto:

Cuando se llama a la función miembro Listen de la clase CAsyncSocket para que el socket comience a escuchar peticiones, si la dirección (compuesta por: la IP o por el nombre de dominio del ordenador que ejecuta Takepicture, más un puerto) está en uso por otra aplicación (por ejemplo, por otra instancia de Takepicture), Listen retornará un 0 para indicar que se ha producido un error, que se podrá saber de qué tipo es accediendo a la función miembro **GetLastError**, la cual retornará, en este caso, un entero asociado a la constante WSAEADDRINUSE.

La modificación propuesta consiste en ejecutar un bucle que vaya ejecutando la función Listen hasta que no retorne ningún error. En caso de retornar error, se ejecutará una nueva iteración del bucle intentando asociar el siguiente puerto al socket. En principio se intenta con el puerto TCP 54321, y se va incrementando en uno por cada iteración del bucle.

```
BOOL CBaseClientDlg::StartListening()
{
	BOOL bOk = FALSE;
	StopListening();
	m_pListen = new CListenSocket(this);
	if ( m_pListen )
	{
		while (!bOk)
		{
			if ( m_pListen->Create( m_uPort, SOCK_STREAM, FD_ACCEPT ) )
				bOk = m_pListen->Listen();

			if ( !bOk )
```

```
                {
                      CString strMsg;
                      int nErr = m_pListen->GetLastError();
                      if ( nErr == WSAEADDRINUSE )
                      {
                            num_aibo ++;
                            strMsg.Format("El puerto %d está en uso, se usará el
%d\n
                                                El  nº  asignado  al  Aibo  controlado
será %d", m_uPort, m_uPort + 1, num_aibo);
                            m_uPort ++;
                            AfxMessageBox(strMsg, MB_OK);
                      }
                }
           }
      }
      else
            AfxMessageBox( IDS_CANT_LISTEN, MB_OK|MB_ICONSTOP );

      return bOk;
}
```

Cada objeto socket se asocia a una dirección IP o a un nombre de dominio en la red. Cuando se quiere crear un socket, normalmente no es necesario especificar esta dirección, ya que es la de la propia máquina donde se crea el socket. Sin embargo, hay que tener en cuenta que es posible que en la máquina haya instaladas varias tarjetas de red (con lo cual habría distintas direcciones IP para la misma máquina) o incluso habiendo una sola tarjeta de red, que haya diferentes direcciones IP asociadas a ella. En estos casos sí sería necesario especificar la dirección al crear un socket, para especificar qué tarjeta de red y/o interficie de red deberá usar el socket, cambiando la línea

```
                if ( m_pListen->Create( m_uPort, SOCK_STREAM, FD_ACCEPT ) )
```

por

```
                if   (   m_pListen->Create(   m_uPort,   SOCK_STREAM,   FD_ACCEPT,
IP_de_la_interficie_de_red ) )
```

2.5.3. Mostrar lo que están viendo los Aibos

Los Aibos pueden enviar imágenes (fotos) en formato jpg al ordenador que los controla (el servidor) gracias a la clase **CImageTP** de la aplicación Takepicture. Los objetos de la clase CBaseClientDlg tienen una variable miembro, **m_pImageTP**, que es un puntero a CImageTP y a través del cual se puede acceder a los métodos y propiedades de los objetos de la clase CImageTP. De esta manera se puede acceder al método TakePicture de CImageTP, al cual se le pasa como parámetro la ruta y nombre del fichero de imagen que se guardará en el disco del servidor.
De la misma manera, CImageTP tiene un puntero como variable miembro que apunta al objeto de la clase CBaseClientDlg para poder acceder a sus funciones miembro.

Takepicture es capaz, mediante un objeto de la clase CImageTP, de ir recibiendo imágenes que el Aibo le envía. Además puede guardar una copia de éstas en disco, lo cual se aprovechará para que el cliente pueda verlas en su navegador.
Desde CBaseClientDlg se accede a los objetos de CImageTP y a los métodos mediante el puntero **m_pImageTP**; desde CImageTP, se accede a los objetos de CBaseClientDlg mediante **m_pDlg**. Al dársele la orden a Takepicture para que empiece a recibir imágenes del Aibo, se crea un objeto de la clase CImageTP.

A través de CImageTP::HandleImageData, se van recibiendo las imágenes que capta el Aibo. Esta función trata la imagen que el Aibo envía de diversas maneras: la muestra en una ventana de Takepicture, y también la guarda en un fichero. Con **ImageSetWindowHandle**, inicializa, en caso de que no esté ya inicializado, un manejador de eventos para la ventana que mostrará automáticamente las imágenes que vayan recibiéndose del Aibo. Con CVAIBO::**ImageGetData**, las va colocando en el búfer CImageTP.**m_imagebuf**.

Lo ideal sería que los Aibos fuesen capaces de enviar un stream de vídeo al servidor, pero no es así, por lo que habrá que hacer que éste construya el stream de vídeo a partir de las imágenes fijas que le

envía el Aibo (opción más difícil), o bien, en lugar de construir un stream de vídeo, que vaya enviando las fotos una a una (opción más fácil).

En un principio se intentó la primera opción (construir el stream a partir de las fotos). La idea es construir un stream en formato **mjpg** a partir de las fotos tomadas en formato **jpg**, y enviarlo a los clientes que se conecten. Como no es un stream que se termine, ya que es un stream de contenido en directo, habría que enviarle al reproductor del cliente una lista de reproducción con la opción de repetir el contenido de la lista activada, para que cuando se terminase de reproducir el stream de mjpg, se volviese a reproducir el nuevo stream de mjpg creado. El servidor tendría que ir creando nuevos streams de mjpg, cada stream con un grupo distinto de fotos tomadas por el Aibo. Sin embargo, aunque con VLC sí se puede enviar streams de mjpg ya creados, parece que no permite crear mjpgs a partir de imágenes jpg.

La segunda opción, que fue la escogida ya que no se pudo crear mjpgs, fue hacer que el navegador del cliente mostrase la imagen vista por cada Aibo haciendo que las imágenes de cada Aibo se viesen en un frame del navegador: si se quiere ver la imagen de lo que ven 4 Aibos, habrá un frame en el navegador para mostrarlas.

Cada frame ha de actualizarse de forma automática cada pocos segundos para actualizar las imágenes, lo cual se consigue con el siguiente código JavaScript:

```
<script>
function recargar(){
      setTimeout("document.form_img1.submit()",5000);
}
</script>
```

document.form_img1 es un formulario que se recarga a sí mismo de forma automática cada 5 segundos (5000 ms).

Cuando llega al servidor web esta petición de recarga, el servidor crea el objeto $**aibo** de la clase **AIBOControl** mediante la sentencia de php `$aibo = new AIBOControl(`*número_de_puerto*`);` que conectará, mediante un socket, a la máquina donde se ejecuta la aplicación que controla al Aibo (Takepicture). En este caso, Takepicture se ejecuta en el mismo servidor web, por lo que el socket se crea de forma local en el propio servidor. Mediante la función de php **fsockopen** se crea el socket que se comunicará con el host especificado en el primer parámetro de fsockopen y le enviará los datos al puerto especificado en el segundo parámetro de fsockopen. Este puerto se indica antes en el constructor de AIBOControl, y según el Aibo que el usuario elija, se especificará un puerto u otro al construir el objeto de AIBOControl. Como ya se ha comentado, se usan los puertos a partir del 54321 en adelante. Por ejemplo, si se pueden controlar 4 Aibos, se usarán los puertos 54321, 54322, 54323 y 54324. Habrá 4 instancias de Takepicture, cada una de las cuales controlará a un Aibo distinto, y cada una de ellas estará escuchando por un puerto distinto: una por el 54321, otra por el 54322, otra por el 54323 y otra por el 54324.

Llegado a este punto, hay 2 alternativas:

Alternativa 1:

La aplicación php del servidor se comunica con la aplicación Takepicture (que puede estar en el mismo servidor o bien en otra máquina) mediante este socket, y al ejecutarse `$ret = $aibo->getImage();` , se accede al método **getImage** del objeto $**aibo**, que envía, mediante el socket, un comando a Takepicture para que guarde, en el disco de la máquina en la que Takepicture se está ejecutando, la última imagen enviada por el Aibo.

Mediante `$ret = $aibo->getImage();` , se accede al método **getImage** del objeto $**aibo**, que retorna una URL donde localizar la imagen. El método **getImage** sería el siguiente:

```
function getImage() {
      $filename = "photo".".jpg";
      $com = "PHOTO"." ".$this->imagepath.$filename;
      $ret = $this->send($com);
      if(strncmp($ret, "OK", 2)==0) {
```

```
            return $this->imageurl.$filename;
      } else {
            return FALSE;
      }
}
```

$**this->imagepath** contiene la ruta de disco donde se guardará la imagen. La variable **imagepath** es miembro de la clase AIBOControl y ya tiene valor asignado nada más crear una instancia de AIBOControl.

Con la línea `$com = "PHOTO"." ".$this->imagepath.$filename;` se crea el comando que le dirá al Takepicture que tome una foto. Se crea concatenando la palabra PHOTO con la ruta de disco donde se guardará la foto, dejando un espacio en blanco entre 'PHOTO' y la ruta, y concatenando a todo ello el nombre del fichero en el que ésta se guardará. Nótese que la concatenación en php se hace con el '.'.

Una vez preparado el comando, se envía al Takepicture con el método **send** de la misma clase AIBOControl. Este método usa la función **fputs**, a la cual se le pasa un descriptor de archivo asociado al socket (que se habrá creado usando el método open previamente y que se comentará más adelante), y el comando, que estará en una cadena de texto. El método **send** es el siguiente:

```
function send($command) {
      if(!$this->fp) {
            return FALSE;
      } else {
            fputs ($this->fp, $command);
            return fgets ($this->fp, 128);
      }
}
```

Después de que el comando haya sido enviado con la función **fputs** de php hacia la aplicación Takepicture a través del socket utilizado para la comunicación bidireccional entre la aplicación Takepicture y la aplicación en php, con **fgets** se recoge un mensaje de texto que habrá sido enviado por Takepicture mediante el mismo socket. Este mensaje es lo que devolverá el método send, e indica

si el comando enviado con anterioridad se ha ejecutado con éxito o no. En caso de éxito, el mensaje contendrá “OK”.

En la función **getImage** se compara este mensaje con la cadena “OK”. Si el mensaje es “OK”, entonces **getImage** retorna la URL de la foto que **Takepicture** habrá guardado en el fichero indicado. Esta URL se especifica concatenando el valor de la variable miembro $**imagepath**, cuyo valor es la ruta del directorio donde se guardará la foto en el servidor web (directorio que ha de colgar del directorio de publicación), con el valor de la variable local $**filename** de **getImage**.
De esta manera, el navegador del cliente podrá acceder a la foto.

La manera de crear el socket que se utiliza para que se comuniquen las aplicaciones php y Takepicture es la siguiente, utilizando el método **open** mencionado con anterioridad:

```
function open() {
      $this->fp = fsockopen($this->host, $this->port, $errno, $errstr, 10);
      if(!$this->fp) {
            return FALSE;
      } else {
            return TRUE;
      }
}
```

El método **open** de AIBOControl se encarga de crear el socket mediante la función de php **fsockopen**, a la cual se pasan como parámetros: la dirección de la máquina donde se ejecuta el Takepicture (que en este caso es la misma donde se estará ejecutando el script php que utilizará a esta clase) con $this->host, el puerto que será usado por la aplicación Takepicture en la máquina en la que se esté ejecutando con $this->port, $errno y $errstr, un entero y una cadena respectivamente, pasados por referencia los dos, que son variables de sistema de php que indican, respectivamente, un código de error y un mensaje de texto del error, y que son actualizadas por fsockopen en caso de error (pueden ser actualizadas por fsockopen porque se pasan por referencia, lo cual se puede saber mirando la cabecera de la declaración de la función fsockopen). Finalmente, el último parámetro (10) es el tiempo

de espera de conexión del socket, que sólo se aplicará para la conexión con el socket y no para la lectura o escritura de datos a través del socket.

Finalmente, para que el explorador del cliente muestre la imagen, el servidor web creará el código HTML necesario de la siguiente manera:

```
echo "<img src=".$ret."><br>";
```

Lo resaltado es para remarcar lo que está entrecomillado.
Lo que va después de la sentencia **echo** es el código HTML que se generará. Lo que está entre comillas, se generará tal cual está, y lo que no lo está tomará el valor de la variable $**ret**, que contendrá la URL de la imagen (la URL que retorna **getImage**). Mediante el punto (‘.’) se concatenan el texto entrecomillado y la URL generada de forma dinámica con la variable $**ret**, y todo ello formará el código HTML que mostrará la imagen.

Alternativa 2:

Como en la primera alternativa, cuando llega al servidor esta petición de recarga, éste crea el objeto $**aibo** de la clase **AIBOControl** mediante la sentencia de php `$aibo = new AIBOControl(54321);` que conectará, mediante un socket, a la máquina donde se ejecuta la aplicación que controla al Aibo. Como la aplicación se ejecuta en el mismo servidor web, el socket se crea de forma local en el propio servidor. Hasta aquí, ocurre lo mismo que en la alternativa 1.

Sin embargo, en la primera alternativa, al ejecutarse `$ret = $aibo->getImage();` en el servidor web, accediendo así al método **getImage** del objeto $**aibo**, se enviaba un comando a Takepicture para que guardase la última imagen enviada por el Aibo.

Resumiendo, en la alternativa 1 ocurre lo siguiente:

1. El explorador del cliente hace petición (al servidor web) de refresco de la imagen que ve un Aibo.
2. Cuando esta petición le llega al servidor web, éste hace una petición a Takepicture para que guarde la foto en disco.
3. Cuando la petición de guardar foto en disco le llega a Takepicture, ésta guarda la foto en disco.

Se puede observar que se van encadenando peticiones en serie, primero desde el cliente al servidor web, y de éste a Takepicture, lo cual no es nada óptimo, ya que un elemento no hace nada mientras no le llegue una petición.

En la alternativa 2, el Takepicture no espera a que lleguen peticiones. El servidor web no enviará la petición de guardar la foto (y por lo tanto Takepicture no esperará esa petición), sino que Takepicture irá guardando (actualizando) la foto cada cierto tiempo, sin esperar petición para hacerlo.

El navegador del cliente, por su parte, irá actualizando el frame del navegador mediante el código en JavaScript explicado anteriormente, enviando la petición de refresco al servidor web (como en la alternativa 1). Cuando la petición llega al servidor web, éste crea entonces la instancia $**aibo** de la clase AIBOControl, y se accede a su método **getImage**, cuyo código ahora es diferente al de la alternativa 1:

```
// Manera nueva, sin que tengan que ir arrastrándose peticiones de un sitio a
otro.
// Simplemente se da por supuesto que el Takepicture ya ha tomado la foto cada x
segundos,
// así que simplemente se accede a ella mediante la URL:
function getImage($num_aibo) {
      $filename = "photo".$num_aibo.".jpg";
      return $this->imageurl.$filename;
}
```

Este método retorna, de la misma manera que en la primera alternativa, la URL de la foto. Obsérvese que, dependiendo de qué Aibo quiera verse la imagen subjetiva (especificado con num_aibo), se retornará la URL de una imagen u otra.

Por su parte, Takepicture, con la clase **CBaseClientDlg** se encarga, básicamente, de manejar los eventos que ocurren en la interfaz de usuario de Takepicture, por ejemplo, si se hace clic en alguno de los botones de la interfaz. Sin embargo, cuando en la página web ocurra algún evento que implique dar alguna instrucción al Aibo, la clases usadas serán **CListenSocket** y **CRequestSocket**, que son las clases que utilizan a los sockets mediante los cuales se reciben las instrucciones que los clientes de la página web quieren dar al Aibo.

La clase CBaseClientDlg tiene una función miembro que se llama **OnTimer**, que es ejecutada cada vez que el timer realiza una interrupción. Para que el timer realice esta interrupción, es necesario activarlo cuando el Takepicture se ejecute, por lo tanto, en la función **CBaseClientDlg::OnInitDialog** se incluye la siguiente línea de código para hacerlo:

```
SetTimer(WM_TIMER, 1*1000, NULL);
```

WM_TIMER es el mensaje asociado a la función de tratamiento de la interrupción: cada vez que el timer provoque una interrupción, se enviará el mensaje WM_TIMER, que será tratado por la función de tratamiento a la interrupción. Esta función se indica en el tercer parámetro con un puntero que apunte a ella, y en este caso es NULL, por lo que se usará la función que hayamos asociado mediante el ClassWizard, que es **CBaseClientDlg::OnTimer**. En el segundo parámetro se indica el número de milisegundos (en este caso 1000) que habrán de pasar para que el timer realice la interrupción.

La función CBaseClientDlg :: OnTimer se encarga, cada vez que es llamada, de hacer una llamada a la función que guarda la foto en el disco, que es la función **CImageTP::TakePicture**, a la cual le pasa como argumento la ruta y nombre del fichero en el que se guardará la imagen.

Cada instancia de Takepicture guardará 1 foto subjetiva, y cada foto tendrá un nombre distinto. Esto se consigue de forma parecida a como se resolvía el problema de asignar un puerto para cada instancia de Takepicture:

En el mismo bucle donde se resuelve el problema de los puertos, en CBaseClientDlg :: StartListening(), se hace lo que está remarcado más oscuro:

```
            while (!bOk)
            {
                  if ( m_pListen->Create( m_uPort, SOCK_STREAM, FD_ACCEPT ) )
                        bOk = m_pListen->Listen();

                  if ( !bOk )
                  {
                        CString strMsg;
                        int nErr = m_pListen->GetLastError();
                        if ( nErr == WSAEADDRINUSE )
                        {
                              num_aibo ++;
                              strMsg.Format("El puerto %d está en uso, se usará el
%d\n
                                                El nº asignado al Aibo controlado será
%d", m_uPort, m_uPort + 1, num_aibo);
                              m_uPort ++;
                              AfxMessageBox(strMsg, MB_OK);
                        }
                  }
            }
```

Antes del bucle, num_aibo está inicializado a 1 en el constructor de CBaseClientDlg.
De esta manera, cada instancia de Takepicture guarda la imagen subjetiva de su Aibo con un nombre distinto: el nombre es igual para todas, excepto en que al final, cada nombre de foto, tiene el número del Aibo al que pertenece: la del Aibo 1 será 'photo1.jpg', la del Aibo 2, 'photo2.jpg', etc.

```
void CBaseClientDlg::OnTimer(UINT nIDEvent)
```

```
{
	if(m_bConnected) {
		// Si hay conexión establecida con un Aibo, hacer y guardar foto en
		// directorio de publicación:
		if (m_pImageTP) {
			CString nombreFichero;
			nombreFichero.Format("%s%d%s",PHOTO_FILE,num_aibo,PHOTO_EXT);
			BOOL bOk = m_pImageTP->TakePicture(PHOTO_FILE);
			if (!bOk) m_nPlayID = -1;
			m_bIsBusy = FALSE;
		}

	}
	// else no hacer foto porque no hay conexión con Aibo
}
```

En la línea remarcada, está la construcción del nombre con el que se guardará la foto. Gracias a la interrupción del timer explicado con anterioridad, se ejecuta esta función, que va actualizando la foto subjetiva del Aibo.

2.5.4. Controlar el movimiento de los Aibos

El usuario puede seleccionar cuál de los Aibos controlar mediante un cuadro desplegable en su navegador. Después de que haya seleccionado uno y después de que se le haya dado su control (más adelante se explicará cómo se le da el control al usuario), si hace clic en una de las flechas del control, el Aibo seleccionado hará el movimiento indicado por la flecha, que puede ser: rotar a la derecha, rotar a la izquierda, caminar hacia adelante o caminar hacia atrás.

Una vez hecho el clic en una de las flechas de control, se envía al servidor web el comando de movimiento del Aibo seleccionado:

```
<a href="#" style="cursor:default"
	onClick="irDireccion('LEFT')"
```

```
      onMouseOver="iluminar('fizq',fizqon.src)"
onMouseOut="iluminar('fizq',fizqoff.src)">
<img src="imagenes/fizqoff.gif" border="0" name="fizq" width="16" height="24">
</a>
```

Al hacer clic en la flecha de rotar a la izquierda, se llama a la función de JavaScript **irDireccion** con el valor 'LEFT' como argumento:

```
function irDireccion(direccion)
{
      document.form_mov.COMMAND.value = direccion;
      resizeTo(60,60);
      document.form_mov.submit();
}
```

El campo oculto COMMAND del formulario toma el valor del comando que será enviado al servidor web.

Una vez le ha llegado el comando al servidor web, éste se encarga de transmitírselo al Aibo al que debe ir dirigido utilizando la clase AIBOControl. Primero crea el objeto $**aibo** de esta clase, inicializando el valor de su variable miembro $**port** con un valor entre 54321 y 54324. Este valor depende del Aibo que el usuario haya escogido. Como se explicó anteriormente, habrá una instancia de Takepicture abierta por cada Aibo que se pueda controlar, y cada una de estas instancias usará un puerto distinto para comunicarse con el script de php. Así, según el Aibo que el usuario haya escogido, $**port** se inicializa con un valor u otro para comunicarse con una u otra instancia de Takepicture.

```
$IDAibo = $_POST["IDAIBO"];  //  Se  recoge  el  valor  del  campo  IDAIBO  del
formulario que controla los Aibos

      [...]
```

```
if($IDAibo == 1)
      $aibo = new AIBOControl(54321);
else if($IDAibo == 2)
      $aibo = new AIBOControl(54322);
else if($IDAibo == 3)
      $aibo = new AIBOControl(54323);
else if($IDAibo == 4)
      $aibo = new AIBOControl(54324);
```

$**IDAibo** tiene un valor que depende del Aibo escogido por el usuario.

Entonces el servidor web crea la comunicación con el socket de la instancia de Takepicture que escuche por este puerto:

```
if($aibo->open()==FALSE) {
      die ("<h1><font color='red'>No se pudo conectar al AIBO</font></h1>");
}
```

y envía el comando a esa instancia de Takepicture:

```
switch($COMMAND) {

      [...]

      case "LEFT":
            $ret = $aibo->left(45);
            break;

      [...]

}
```

La instancia de Takepicture que esté escuchando por ese puerto será la única instancia que recibirá el comando y que actuará en consecuencia (enviando el comando adecuado al Aibo).

Para comunicarse con la aplicación php mediante sockets, Takepicture utiliza dos clases que derivan de **CAsyncSocket**: **CListenSocket** y **CRequestSocket**.
Para comunicarse con el Aibo, Takepicture usa la clase **CVAibo** del Remote Framework.

CListenSocket se utiliza para escuchar peticiones que realiza el servidor web con php, y CRequestSocket para tratarlas.

En CListenSocket, la función miembro **OnAccept** es una función que en CAsyncSocket (clase de la cual derivan CListenSocket y CRequestSocket) está declarada como virtual, y se define en CListenSocket para indicarle, al socket que está escuchando, que tiene peticiones de conexión pendientes y que debe aceptarlas. Es decir, cuando haya peticiones de conexión pendientes en la cola del socket (peticiones de conexión que son iniciadas cuando la aplicación php envía el comando, a través del socket, a Takepicture), se ejecutará **CListenSocket::OnAccept**.

```
void CListenSocket::OnAccept( int /*nErrorCode*/ )
{
	m_pRequest = new CRequestSocket(m_pDlg);
	if ( Accept( *m_pRequest ) )
	{
		m_pRequest->AsyncSelect( FD_READ|FD_CLOSE );
	}
}
```

La función miembro **Accept**, heredada de CAsyncSocket, en la sentencia **Accept(*m_pRequest)** extrae la primera conexión pendiente que haya en la cola de conexiones pendientes y crea un nuevo socket copia en **m_pRequest**, que es un socket de la clase CRequestSocket.

Con **m_pRequest->AsyncSelect(FD_READ|FD_CLOSE);** se solicita que se le notifiquen, al socket copia creado, dos tipos de evento: el de que ya puede leer y el de que ha sido cerrado.

En CAsyncSocket existe la declaración de la función virtual **OnReceive**, que se define en la clase derivada de CAsyncSocket en TakePicture, la clase **CRequestSocket**. La función OnReceive se ejecutará cuando el socket copia creado tenga datos disponibles que pueda recoger con **Receive** (función miembro heredada de CAsyncSocket). Estos datos se guardarán en el búfer **m_buf**, que es una variable miembro tipo array de CRequestSocket.

ProcessCommand lee los datos que han sido colocados en el búfer m_buf. Estos datos están formados por el **comando** y posiblemente por un **parámetro** que acompañará a éste, dependiendo del tipo de comando del que se trate. Exceptuando el comando STOP, los demás comandos que pueden ser enviados desde la aplicación web estarán acompañados de un parámetro: RIGHT y LEFT vendrán seguidos por los grados que habrá de girar el Aibo, y FORWARD y BACK por los centímetros que habrá de caminar, hacia adelante o hacia atrás.

ProcessCommand ha de acceder a los métodos de la instancia de la clase CBaseClientDlg, que es la clase a través de la cual sea accede a los métodos de la clase CVAIBO, la clase del Remote Framework que se utiliza para controlar al Aibo. Lo hace a través del puntero **m_pDlg**, que es variable miembro de la instancia de la clase a la que pertenece ProcessCommand (CRequestSocket).

Cuando la instancia de CBaseClientDlg crea el socket de escucha con CListenSocket, le pasa al constructor de CListenSocket este puntero, **m_pDlg**, que apunta a la propia instancia de CBaseClientDlg. Este puntero no lo usará directamente la instancia de CListenSocket, ésta únicamente se lo pasará a la instancia de CRequestSocket que crea para crear el socket copia que trata las conexiones pendientes.

Cuando el servidor web con php cierra la conexión con el socket de Takepicture de la *instancia de CRequestSocket* (el socket de tratamiento de las conexiones pendientes), automáticamente se llama en Takepicture a la función **OnClose**, que es miembro de la clase CRequestSocket y es una redefinición de la función OnClose de la clase CAsyncSocket (es decir, que en CAsyncSocket es una función virtual). Esta función libera la memoria que utiliza la *instancia de CRequestSocket*.
El servidor web con php cierra la conexión con el socket con `$aibo->close();` antes de finalizar la ejecución del script php que crea la página para el cliente (recordar que $aibo es un puntero que sirve para acceder a las funciones miembro de AIBOControl). Por tanto, se abrirá y cerrará una conexión con cada envío de comando.

Cada función miembro de CBaseClient que envía un comando al Aibo, guarda en la variable miembro **m_nPlayID** el command_ID del comando enviado al Aibo. El comando es enviado sin esperar a su resultado (sin esperar a ver si tiene éxito o no).
La variable miembro **m_bIsBusy** se pone a TRUE cada vez que se envía un comando al Aibo para indicar que el Aibo está ocupado ejecutando el comando.
En caso de que éste no haya sido ejecutado con éxito por el Aibo, sale un mensaje informando de ello en el Takepicture.

2.5.5. Cómo se otorga el control de un Aibo al usuario

Como no puede haber un Aibo exclusivamente para cada usuario que se conecte a la página web, se hace necesario disponer de un mecanismo que vaya dando el control de los Aibos por turnos.
Para controlar **un** Aibo, se dispondrá de **una** cola de peticiones de control. El servidor web irá colocando en esta cola las peticiones para controlar este Aibo que hagan los usuarios. Esta cola tendrá que existir en todo momento mientras se permita conectar a la página web.

Cuando un cliente se conecta a la página web, en el servidor se crea un proceso nuevo que ejecuta el script php que genera la página, de modo que puede haber varios procesos, uno por cliente, que estén siendo ejecutados. Si estos clientes están pidiendo el control del mismo Aibo, cada uno de esos procesos debe colocar, de alguna manera, una petición en la cola. La cola es una estructura que se

encuentra en una determinada zona de memoria (es un array), por lo que se podría hacer que cada uno de estos procesos pudiera acceder a esa zona de memoria y colocar en la cola su petición. Esto hace intuir la posible utilización de semáforos para acceder a una zona de memoria compartida (la cola). Cada proceso debería poder escribir la petición en la cola, y también debería poder leer de ella para saber en qué posición de la cola se encuentra su petición y saber así si ya puede tomar el control o bien tiene que esperar.
Un proceso, antes de leer, debería esperar a poder entrar en su sección crítica (o sea, la sección de su código que lee la cola), es decir, que debería esperar a que se le pusiera su semáforo en verde, por si hay otro proceso en ese momento que está escribiendo en la cola; un proceso, antes de escribir su petición en la cola (es decir, antes de entrar en su sección crítica para modificar la zona de memoria compartida, esto es, la cola), debería poner los semáforos del resto de procesos en rojo, y ponerlos en verde una vez hubiera finalizado de escribir.

Sin embargo, en lugar de que cada proceso de cliente haga su propia gestión de la cola, como implicaría con la utilización de semáforos y memoria compartida, puede ser un único proceso el que se encargue de gestionar las peticiones que le lleguen de los procesos de los clientes. Al ser un único proceso el que se encarga de esta gestión, es decir, sólo habría un **proceso gestor de la cola** que escribiría y/o leería de la cola, ya no habría memoria compartida por varios procesos y por tanto ya no sería necesaria la utilización de semáforos. Por su mayor simplicidad respecto a la de memoria compartida, ésta ha sido la solución escogida.

Ahora, lo único que necesita el proceso de un cliente, es la posibilidad de comunicarse con el proceso gestor de la cola para pedirle que coloque su petición de control en la cola. Esta comunicación es posible mediante el mecanismo de sockets de que dispone php.

El proceso gestor de la cola ha de estar activo en todo momento mientras se sirva la página web, para que los procesos de los clientes puedan realizar peticiones de control y éstas puedan ser colocadas en la cola. El proceso gestor, gracias a los sockets, dispone de una cola de **conexiones** (conexiones de los procesos de los clientes con el proceso gestor) en la que el mecanismo de los sockets va colocando las peticiones de conexión que realicen los procesos de los clientes. Cada una de estas peticiones de conexión será procesada (es decir, se meterá una petición para ese proceso cliente en la **cola de peticiones**). Mientras el proceso gestor esté haciendo una gestión con una petición (o sea, mientras esté colocándola en la cola de peticiones), podrán ir llegando peticiones de conexión a la cola del

socket, y cuando el proceso gestor ya haya terminado de realizar esa gestión, admitirá la siguiente conexión de la cola de conexiones.

Se puede observar la similitud de comportamiento entre el proceso gestor con su cola de conexiones y la primera solución propuesta con semáforos y memoria compartida: cada conexión colocada en la cola de conexiones que tenga que esperar a ser procesada se comporta de forma similar a un proceso de cliente que tenga que esperar a que se le ponga su semáforo en verde. De hecho, con la solución del proceso gestor, los procesos de cliente deben esperar a que el proceso gestor admita su petición de conexión, con lo que el comportamiento es similar al que se tendría con semáforos y memoria compartida, pero al ser este comportamiento intrínseco a los sockets, la simplicidad es mayor, ya que aprovechamos que los sockets tienen de forma interna este mecanismo que es similar al de los semáforos.

Lo que llamamos petición, a la que el proceso gestor coloca en la cola de peticiones, es en realidad el **session_id** (identificador de sesión) del cliente que realiza la petición. A cada cliente que se conecta al servidor web, el proceso de cliente le asigna un **session_id** único, para lo cual se utiliza la función de php **session_start()** al inicio del script ejecutado por el proceso de cliente, que asigna un nuevo session_id a un cliente nuevo que se conecte, o bien mantiene el mismo session_id de un cliente que ya estaba conectado con el servidor web.

El proceso gestor no sólo se encarga de colocar peticiones en la cola de peticiones de control, también le dice al proceso de cliente que lo requiera cuánto tiempo deberá esperar ese cliente hasta poder controlar el Aibo.

En realidad, no hay sólo una cola de peticiones de control, sino tantas como Aibos diferentes puedan controlarse, o sea, hay una cola de petición de control por cada Aibo. Un cliente podrá elegir entre pedir el control de un Aibo concreto, o bien pedir el control de uno cualquiera de los Aibos. Si el cliente pide el control de un Aibo concreto, el proceso gestor colocará su petición en la cola asociada a ese Aibo; si el cliente pide el control de uno cualquiera de los Aibos, el proceso gestor colocará su petición en una de las colas que tengan menos peticiones (el proceso gestor la elegirá).

Cuando un cliente pide el control del Aibo, su proceso de cliente que se ejecuta en el servidor web pide al proceso gestor de las colas que meta su session_id en una cola (puede ser una cola concreta

elegida por el cliente o elegida por el proceso gestor). El proceso gestor responde entonces al proceso del cliente, enviándole el número de segundos que ese cliente habrá de esperar hasta que su session_id llegue a la cabeza de la cola elegida. Si el session_id del cliente ya está en la cabeza de la cola, el proceso le enviará un 0 para indicarle que ya puede controlar el Aibo asociado a esa cola. De esta manera, el proceso de cliente podrá generar la página para el cliente haciendo que se muestre en el navegador del cliente cuántos segundos le quedan de espera para tomar el control del Aibo.

El usuario cuyo session_id se encuentre en la cabeza de una cola, es el que tiene el control del Aibo asociado a esa cola. Si hay más session_ids en esa cola (o sea, más usuarios esperando para controlar el Aibo asociado a esa cola) es necesario que el session_id de la cabeza esté en ésta por un tiempo limitado, y pasado ese tiempo se elimine de la cola para que ésta avance. Si por el contrario hay sólo una petición, (no habría más usuarios esperando a tener el control del Aibo asociado a esa cola), no tendría sentido quitar el control del Aibo al cliente con el session_id de la cabeza, por ello, si en una cola sólo hay un session_id, se le deja en la cola por un tiempo indefinido, y sólo se pondrá en marcha su contador de tiempo para quitarlo de la cola si llega otro session_id a ésta.

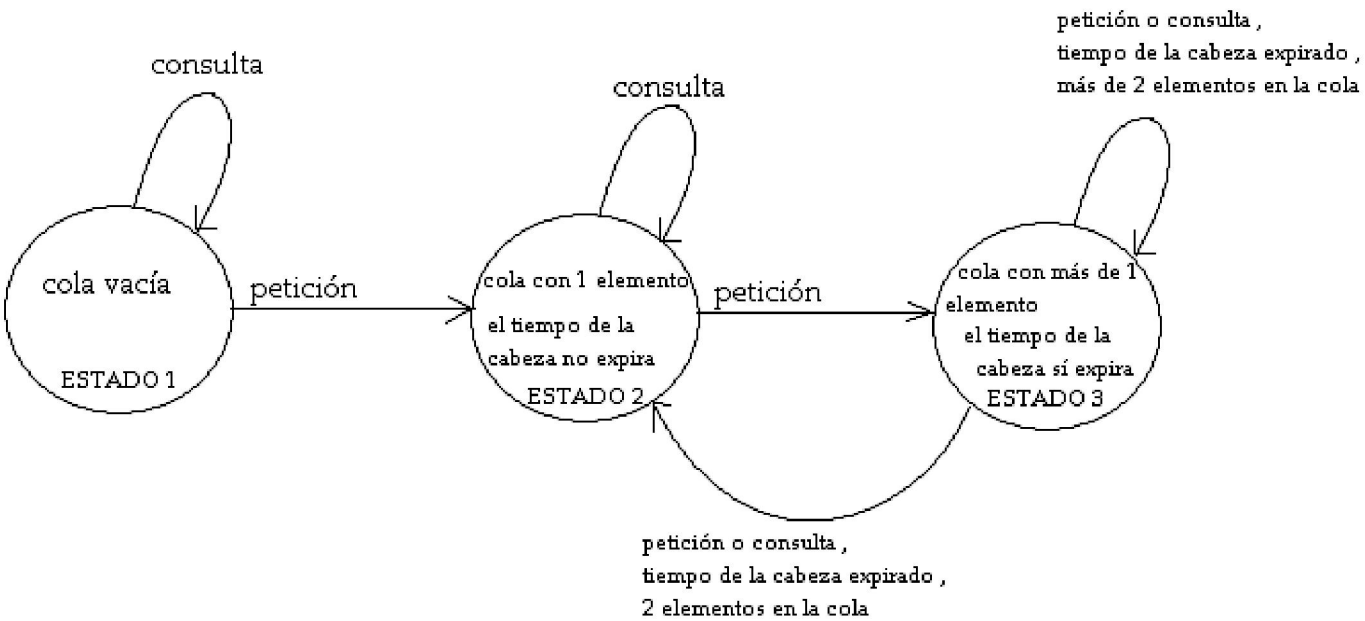
Cada una de las colas es un objeto de la clase **ColaPeticiones**. El proceso gestor crea tantos de estos objetos como Aibos se puedan controlar. Un objeto cola tiene 4 variables miembro:
$cola es el array donde se irán poniendo los session_ids, que se comportará como una cola de peticiones como las descritas antes;
$hora es la hora en segundos desde el Epoch Unix (1 de enero de 1970 a las 00:00:00 GMT) a la que el sesion_id de la cabeza llegó a ésta, y sirve como referencia para saber cuánto tiempo lleva el session_id de la cabeza de la cola en la cabeza de la cola (o sea, cuánto tiempo lleva teniendo el control del Aibo el usuario cuyo session_id es el de la cabeza). Normalmente, se actualiza cada vez que un session_id alcanza la cabeza de la cola, esto es cuando la cola avanza, y también cuando la cola pasa de tener un solo session_id a tener más; pero si un session_id alcanza la cabeza de la cola por estar ésta vacía, es decir, que estando la cola vacía llega el session_id (y por lo tanto llega a la cabeza nada más entrar en la cola), entonces no es necesario actualizarlo, porque no se tomará como referencia, ya que no se necesita saber cuánto tiempo lleva en la cabeza un session_id mientras esté solo en la cola, debido a que no su tiempo no expira mientras no lleguen más session_ids. Si en esta situación llega entonces otro session_id a la cola, entonces sí se actualizará, ya que habrá de expirar el control del usuario cuyo session_id es el de la cabeza. Lo mismo ocurre en caso de que después de

avanzar la cola, quede sólo un session_id en ella: como sólo queda un session_id en la cola, se mantendrá ahí de forma indefinida hasta que lleguen más session_ids, por lo que no será necesario actualizar $hora.

$limite es la cantidad de segundos que el cliente cuyo session_id es el de la cabeza de la cola tendrá el control del Aibo. Siempre que el proceso que gestiona las peticiones de control recibe algún tipo de instrucción, ya sea para meter un nuevo session_id en una cola o para que retorne el número de segundos que un cliente ha de esperar, resta de la hora actual *$hora* segundos, para comparar el resultado con *$limite* y ver así si ha de quitarse de la cabeza el session_id actual. Lo normal es que el valor de $limite sea *$limitePorDefecto*, que es una variable miembro de ColaPeticiones que tiene un valor constante, pero si en algún momento la cola se queda con un solo session_id, entonces $limite toma como valor un 0, lo cual le indica a la función miembro que se encarga de que la cola avance, que no debe avanzar.

Puede observarse así que, aunque nada más ser lanzado el proceso todas las colas estarán vacías, una vez ha entrado un session_id en una cola, esa cola ya nunca volverá a estar vacía, siempre tendrá al menos un session_id.

Grafo del comportamiento de la cola:



- Petición significa que llega una petición de un cliente para poner su session_id en la cola.
- Consulta significa que un cliente quiere conocer el tiempo que le queda para llegar a la cabeza de la cola.

- Después de pasar del estado 1 al 2, se añade un session_id a la cola.
- Después de pasar del estado 2 al 3, se añade un session_id a la cola.
- Después de pasar del estado 3 al 2, se quita un session_id de la cola.
- Después de pasar del estado 3 al estado 3 de nuevo, se quita un session_id de la cola.

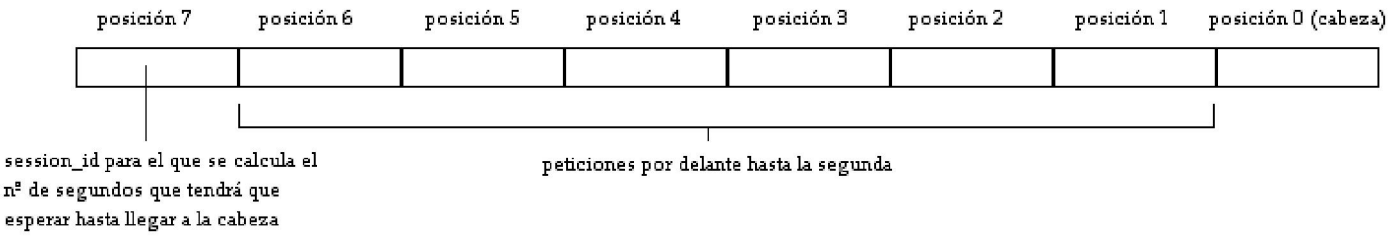
El proceso que gestiona las colas, mientras no recibe peticiones de conexión por el socket, no hace nada más que escuchar si hay peticiones de conexión. Cuando en la cola de conexiones hay peticiones de conexión, es cuando se pone en funcionamiento la gestión de las colas. Esto es así porque el socket utilizado por el proceso para escuchar peticiones de conexión es bloqueante, por lo que la ejecución del programa no continúa mientras no llegan nuevas peticiones de conexión. Esto hace que el consumo de CPU sea mucho menor de lo que sería con un socket no bloqueante, y además nos basta con que el socket sea bloqueante porque no necesitamos que el proceso esté continuamente mirando si tiene que avanzar colas y quitando session_ids de la cabeza de las colas.

Cómo calcular el tiempo que le queda a un usuario para tomar el control de un Aibo

Cuando la petición de un cliente se mete en una cola, el proceso gestor de las colas retorna al proceso del usuario el número de segundos que ese cliente tendrá que esperar hasta poder controlar el Aibo. También se retorna este número al proceso del cliente si así lo solicita éste después.
Este número se calcula en función de la posición que ocupe la petición del cliente (hablamos indistintamente de peticiones o de session_ids) en la cola. Este cliente habrá de esperar tantos segundos como tarde su petición en alcanzar la cabeza de la cola, es decir, que habrá que calcular cuánto tiempo total estarán en la cabeza todas las peticiones que le preceden. Este tiempo es el número de segundos que le queden a la petición de la cabeza, más el número de segundos que vayan a estar en total el resto de peticiones precedentes a la del cliente.

El número de segundos que le quedan a la petición de la cabeza es *$limite* - *(time() - $hora)*, donde *time()* es una función de php que retorna la hora actual medida en número de segundos desde el Epoch Unix (1 de enero de 1970 a las 00:00:00 GMT), y *(time() - $hora)* es el número de segundos que lleva en la cabeza la petición que hay en ella.

La cantidad de segundos que estarán en total entre todas las demás peticiones es el *nº de peticiones por delante hasta la segunda * $limite*, ya que todas esas peticiones estarán el mismo tiempo (*$limite* segundos) en la cabeza de la cola.



El *nº de peticiones por delante hasta la segunda* es *posición_del_session_id - 1*, así que el tiempo total que estarán en la cabeza todas las peticiones por delante hasta la segunda es (*posición_del_session_id - 1) * $limite*, con lo que la fórmula final para calcular el tiempo de espera de un session_id es:
$hora + $limite - time() + (posición_del_session_id - 1) * $limite.

De esta manera, el proceso del usuario puede crear, además del código html para el navegador del usuario, el código javascript que haga que la página se recargue pasado ese número de segundos y vuelva así a ejecutarse un nuevo proceso de usuario para ese usuario.

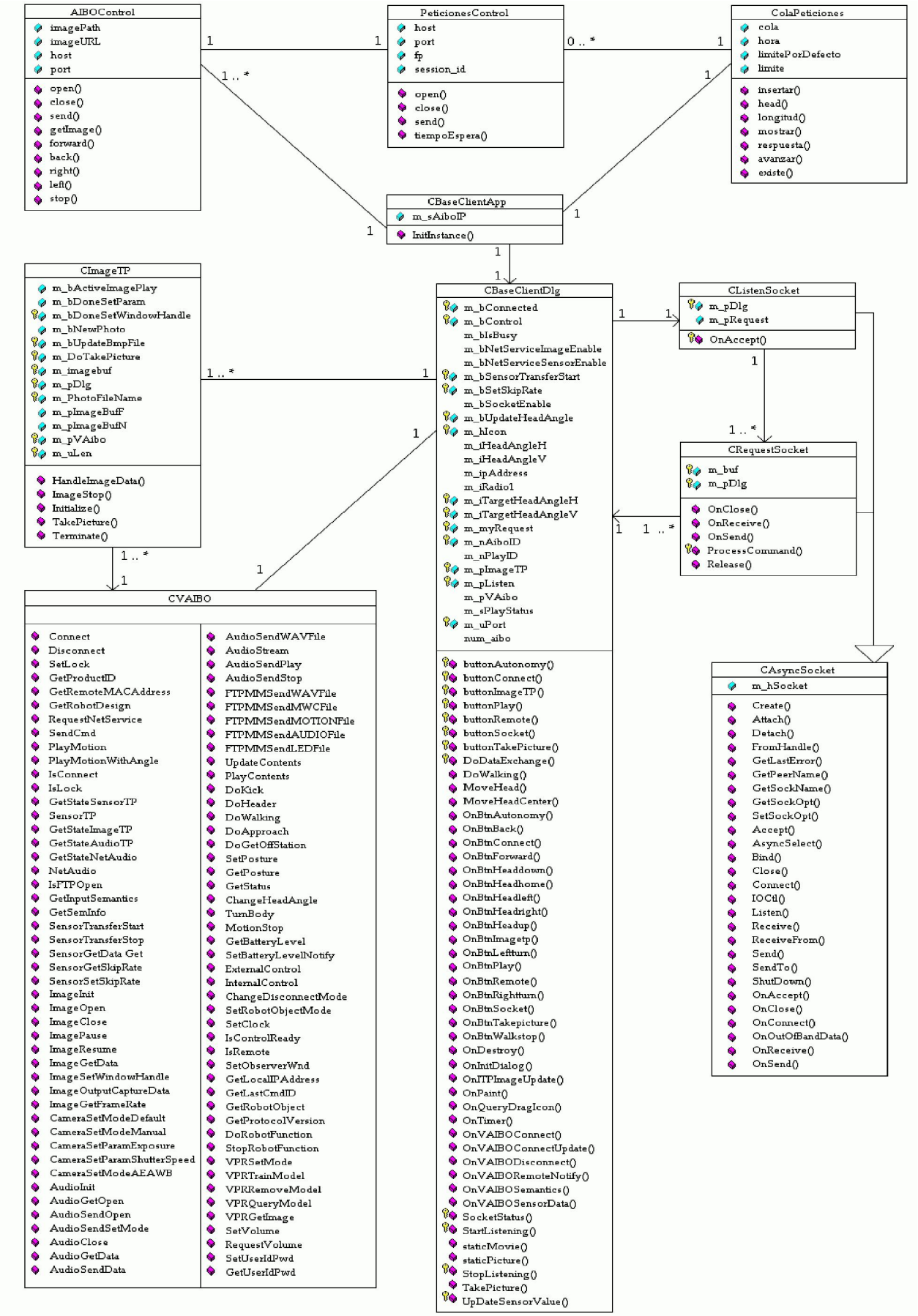
Podría darse el caso de que uno o más procesos de los usuarios que están por delante en la cola, no se ejecutasen: si un usuario cierra su navegador, éste no podrá entonces hacer que se recargue la página, con lo que no se ejecutará el proceso de este usuario en el servidor, y al no ejecutarse, no enviará al proceso que gestiona las colas la consulta para saber cuántos segundos debe esperar su cliente. Al no llegarle ninguna consulta ni petición al proceso que gestiona las colas (y si tampoco le llega alguna consulta o petición de algún otro cliente), éste no avanzará en su ejecución y por lo tanto no irá quitando de las colas las peticiones a las que ya se les haya pasado su turno.

Por este motivo es necesario un mecanismo que no sólo quite de la cabeza de la cola la petición que haya expirado, sino también el resto de peticiones que le sigan que también hayan expirado. Es decir, que habrán de quitarse de la cola todas las peticiones que ya hayan expirado, de una sola vez.

Si $hora + $limite - time() es positivo, indica cuánto tiempo de control le queda al primero de la cola, y si es negativo indica cuántos segundos hace que el primero dejó de tener el control del Aibo. Este número negativo se puede aprovechar para quitar de la cola a los que ya hayan expirado, de la siguiente manera:

- Se pasa este número a positivo.
- Se calcula *posicion := número_en_positivo* div $limite y *segundos := número_en_positivo* mod $limite , donde div es la división entera y mod es el módulo. *Posicion* es el número de la posición de la cola a partir del que hay que eliminar peticiones, ya que éstas han expirado. *Segundos* indica cuántos segundos hace que se le pasó el tiempo a la petición que está en la posición *posicion*.

2.5.6. Diagrama de clases



3. Conclusiones

Básicamente se han conseguido los objetivos que se pretendía conseguir:

- La utilización de un software capaz de servir streams en directo, aunque sería posible añadir mejoras, como el multicast, haciendo que la transmisión fuese parecida a la de una cadena de televisión, y como el RTP (Real-Time Transport Protocol), haciendo que las imágenes del stream al usuario le llegasen con el menor retraso posible.

- Vista de las imágenes subjetivas de los Aibos, aunque también se podría mejorar de la misma forma que los streams en directo.

- La posibilidad de controlar Aibos por turnos a través de Internet.

4. Referencias bibliográficas

http://www.videolan.org/doc/streaming-howto/en/streaming-howto-en.html

http://www.videolan.org/doc/play-howto/en/play-howto-en.html

http://forum.videolan.org

http://developer.apple.com/opensource/server/streaming/index.html

http://www.rediris.es/rediris/boletin/58-59/ponencia10.html

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwmt/html/adding_windows_media_to_web_pages__etse.asp

http://jungla.dit.upm.es/~jmseyas/linux/mcast.como/Multicast-Como.html

http://www.stream-video.com/FQstream.htm

http://www.soundscreen.com/streaming/embed_streams.html

http://www.php.net

http://www.w3schools.com

http://www.desarrolloweb.com

http://www.linuxtoday.com/news_story.php3?ltsn=2001-07-10-003-20-PR-KE-MS

Documentación MSDN Visual Studio 6

Autor: José Manuel Cañete Poyatos
Títol: Control remot (web) d'un robot Aibo
Any d'elaboració: 2006
Director: Jordi Vitrià Marca
Tipus de treball: Projecte de fi de carrera
Titulació: Enginyeria en informàtica
Escola/Facultat: Escola Tècnica Superior d'Enginyeria
Paraules clau:
control, remoto, web, robot, aibo, memoria, proyecto
control, remot, web, robot, aibo, memòria, projecte
Resum:

El robot Aibo dispone de la librería Aibo Remote Framework para controlarlo remotamente mediante un PC y una red inalámbrica, también para acceder a información de estado del robot, o para ver las imágenes que éste capta.
Combinando Remote Framework y php se ha creado una aplicación web que permite controlar varios Aibos remotamente por Internet, así como tener acceso a las imágenes subjetivas de cada uno de los Aibos.
Además, la tecnología existente de streaming permite a la aplicación web tener un vídeo incrustado que permite ver en directo los Aibos mediante una cámara web enfocada hacia ellos.

El robot Aibo disposa de la llibreria Aibo Remote Framework per controlar-lo remotament mitjançant un PC i una xarxa inalàmbrica, també per accedir a informació d'estat del robot, o per veure les imatges que l'Aibo capta.
Combinant Remote Framework i php s'ha creat una aplicació web que permet controlar Aibos diferents remotament per Internet, així com tenir accés a les imatges subjectives de cadascun dels Aibos.
A més, la tecnologia existent d'streaming permet que l'aplicació web tingui un vídeo incrustat que permet veure en directe els Aibos mitjançant una càmara web enfocada cap a ells.