5. Aplicación desarrollada

En este capítulo veremos la aplicación desarrollada (en C#) para aprovechar una de las funcionalidades del Adaptador de Red GC-100 y aportaremos un manual para los futuros usuarios de este sistema.

Una de las principales funcionalidades que aporta el GC-100 a una red domótica es la capacidad para recibir y transmitir señales electromagnéticas del rango de los infrarrojos (IR). Estas señales son usadas por la mayoría de dispositivos de control remoto de los aparatos electrónicos caseros.

Nos proponemos desarrollar una aplicación que permita a una red domótica controlar cualquier dispositivo electrónico que cuente con un control remoto simulando a este último gracias al Adaptador de Red GC-100.

En primer lugar creamos una aplicación capaz de guardar los mensajes asociados a cada orden enviada por un mando de control. A cada orden IR registrada por el GC-100 (que la recibe gracias al GC-IRL conectado a uno de sus puertos serie) pedirá al usuario un nombre al que asociarla. Estas parejas formadas por los nombres insertados por el usuario y los códigos IR recibidos a través del GC-IRL se guardarán en un archivo XML. Posteriormente podremos crear un control remoto "virtual" en nuestro PC leyendo cualquiera de los archivos XML que hayamos guardado previamente. La tercera funcionalidad de nuestro programa permitirá al sistema reconocer una orden IR de un control remoto guardada previamente y retransmitirla por cualquiera de sus salidas.

En segundo lugar, adaptamos el programa creado a la arquitectura UPnP (Universal Plug and Play). Esto permitirá que una red domótica con un punto de control UPnP integrado interactúe con nuestro programa tratándolo como un dispositivo más de la red.

Escribimos la aplicación en el lenguaje de programación C# y usamos el entorno de desarrollo Microsoft Visual Studio 2005.

5.1 Descripción del código

5.1.1 Recepción y transmisión de códigos infrarrojos

Lo primero que debemos tener en cuenta es que todo nuestro programa girará en torno al Adaptador de Red GC-100, por tanto, tendremos que establecer comunicación con él. Como ya sabemos, el GC-100 tiene una conexión RJ-45 con la que puede conectarse a una red Ethernet mediante el correspondiente cable de pares. Para que un sistema se comunique con él deberá conocer su dirección IP y el número de Puerto TCP adecuado. Nos familiarizaremos con el concepto de socket.

Socket designa un concepto abstracto por el cual dos programas (posiblemente situados en sistemas distintos) pueden intercambiarse cualquier flujo de datos, generalmente de manera fiable y ordenada. Un socket queda definido por una dirección IP, un protocolo y un número de puerto.

Haremos uso de la clase *Socket* que nos proporciona la plataforma *.NET*. La clase *Socket* proporciona un extenso conjunto de métodos y propiedades para las comunicaciones en red. La clase *Socket* permite realizar transferencias de datos sincrónicas y asincrónicas mediante cualquiera de los protocolos de comunicación incluidos en la enumeración *ProtocolType*. Para nuestro proyecto nos interesan las transferencias asíncronas mediante el protocolo de trasporte TCP.

Con el fin de procesar las comunicaciones mediante diferentes subprocesos durante la ejecución, utilizaremos los métodos *BeginConnect* y *EndConnect* para conectar con un puerto de escucha. Usaremos los métodos *BeginSend* y *EndSend* o *BeginReceive* y *EndReceive* para comunicar datos asincrónicamente. Si realizamos varias operaciones asincrónicas en un socket, no tienen que completarse necesariamente en el orden en que se inician.

Cuando terminemos de enviar y recibir datos, usaremos el método *Shutdown* para deshabilitar *Socket*. Después de llamar a *Shutdown*, llamaremos al método *Close* para liberar todos los recursos asociados al *Socket*.

El programa va a estar formado por un formulario principal que dará acceso a otros tres formularios que recogerán cada una de las funcionalidades aportadas por la aplicación.

RecibeIR

Este formulario dará al usuario la posibilidad de almacenar códigos de control infrarrojo en el sistema para poder utilizarlos posteriormente. El usuario introducirá el nombre que desea dar a una orden (por ejemplo "Encender") y, a continuación, pulsará el botón correspondiente del mando a distancia con éste apuntando directamente al receptor IR del GC-IRL. El GC-IRL debe estar conectado a uno de los puertos serie del GC-100. Una vez introducidos todos los comandos deseados estos se guardarán en un archivo XML.

En primer lugar hay que establecer la conexión mediante el Socket con el GC-100:

```
private void bt_Conectar_Click(object sender, EventArgs e)
      {
         try
         {
            if (!tb_IP.Text.Equals("") && (tb_Puerto.Text.Equals("4999") ||
tb_Puerto.Text.Equals("5000")))
             {
                bt_Conectar.Enabled = false;
                tb_IP.Enabled = false;
                lb_IP.Enabled = false;
                tb_Puerto.Enabled = false;
                lb_Puerto.Enabled = false;

                conectado = false;
                cliente = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
            //Crea el Socket
                IPEndPoint ipep = new IPEndPoint(IPAddress.Parse(tb_IP.Text), int.Parse(tb_Puerto.Text));
            //IP y puerto TCP
                cliente.BeginConnect(ipep, new AsyncCallback(connectCallback), cliente);
            //Comienza solicitud asíncrona

                System.Threading.Thread.Sleep(1000);
                if (conectado)
                {
                   ArchivoXML.guardarIPpuertoXML(tb_IP.Text, tb_Puerto.Text);
                   bt_Desconectar.Enabled = true;
                   bt_Crear.Enabled = true;
                   lb_texto.Enabled = true;
                   lb_texto.Text = "Nombre del fichero XML";
                   tb_texto.Enabled = true;
                   toolStripStatusLabel1.Text = "Introduzca el nombre del fichero XML (sin extensión)";
                }
```

```
                else
                {
                    bt_Conectar.Enabled = true;
                    tb_IP.Enabled = true;
                    lb_IP.Enabled = true;
                    tb_Puerto.Enabled = true;
                    lb_Puerto.Enabled = true;
                    toolStripStatusLabel1.Text = "La conexión no se estableció, compruebe la IP, el puerto y
los elementos de su sistema";
                }
            }
            else
            {
                toolStripStatusLabel1.Text = "Escribe la IP y el puerto (4999 o 5000)";
            }
        }
        catch (SocketException)
        {
            toolStripStatusLabel1.Text = "La conexión no es posible";
            bt_Conectar.Enabled = true;
            tb_IP.Enabled = true;
            lb_IP.Enabled = true;
            tb_Puerto.Enabled = true;
            lb_Puerto.Enabled = true;
        }
    }
```

Mediante la orden:

cliente.BeginConnect(ipep, new AsyncCallback(connectCallback), cliente);

Se solicita la conexión asíncrona a la dirección IP y puerto TCP recogidos por *ipep*. El delegado de *AsyncCallback* hace referencia al método de devolución de llamada al que se llamará cuando haya finalizado la operación asincrónica.

```
//Método llamado cuando finaliza la solicitud asíncrona de conexión
private void connectCallback(IAsyncResult ar)
    {
        try
        {
            cliente = (Socket)ar.AsyncState;
            cliente.EndConnect(ar);
            conectado = true;
        }
        catch (Exception)
        {}
    }
```

La variable *conectado* nos sirve para comprobar que el método de devolución de llamada se ha ejecutado correctamente, es decir, que la conexión se ha establecido.

Con la conexión establecida el usuario podrá pasar a registrar los códigos IR del control remoto elegido. El programa guardará cada uno de ellos en una matriz de cadenas de caracteres. La primera cadena almacenará el nombre con el que el usuario quiere reconocer el archivo XML que se creará al finalizar. Cada una de las siguientes cadenas comenzará con el nombre dado por el usuario para cada comando IR seguido por el carácter ';'. A continuación el programa añade el código recibido del GC-100.

```
private void bt_Aceptar_Click(object sender, EventArgs e)
      {
         int indice;
         bool repetido;
         try
         {
            repetido = false;
            if (i < instrucciones.Length)
            {
               if (!tb_texto.Text.Equals(string.Empty) || !instrucciones[i].Equals(string.Empty))
               {
                  for (indice = 1; indice < i; indice++)
                  {
                     if
(tb_texto.Text.Equals(instrucciones[indice].Substring(0,instrucciones[indice].IndexOf(';'))))
                     {
                        repetido = true;
                     }
                  }
                  if (repetido)
                  {
                     toolStripStatusLabel1.Text = "Ya ha guardado un comando con ese nombre, introduzca
otro por favor";
                  }
                  else
                  {
                     if (instrucciones[i].Equals(string.Empty))
                     {
                        instrucciones[i] = tb_texto.Text + ";";
                        toolStripStatusLabel1.Text = "Acerque el control remoto al receptor y pulse la tecla
del comando \"" + instrucciones[i].Substring(0, instrucciones[i].IndexOf(';')) + "\"";
                        //Comienza a recibir asincrónicamente los datos de un objeto Socket conectado
                        cliente.BeginReceive(buffer, 0, 255, SocketFlags.None, new
AsyncCallback(receiveCallback), cliente);
                     }
                     else
                     {
                        toolStripStatusLabel1.Text = "Esperando el comando IR \"" +
instrucciones[i].Substring(0, instrucciones[i].IndexOf(';')) + "\"";
                     }
                  }
               }
               else
               {
                  toolStripStatusLabel1.Text = "Debe introducir un nombre para el comando IR";
               }
            }
            else
```

```
            {
                toolStripStatusLabel1.Text = "Se ha llegado al máximo de comandos IR para guardar";
            }

            tb_texto.Text = string.Empty;
        }
        catch (Exception)
        {
            toolStripStatusLabel1.Text = "Ocurrió un error inesperado mientras introducía el nombre de un
comando";
        }
    }
```

Tras introducir un nombre para identificar la instrucción IR a guardar, el programa ejecuta la orden:

cliente.BeginReceive(buffer, 0, 255, SocketFlags.None, new AsyncCallback(receiveCallback), cliente);

De este modo se pone a la espera de recibir asíncronamente datos del Socket conectado con el puerto correspondiente (aquél al que está conectado el GC-IRL) del GC-100. Tras finalizar la recepción se ejecuta el método de devolución de llamada *receiveCallback.*

```
private void receiveCallback(IAsyncResult ar)
    {
        int indice;
        int bytesLeidos;
        string cadena;
        bool ok;
        try
        {
            bytesLeidos = cliente.EndReceive(ar);
            cadena = string.Empty;
            cadena = Encoding.ASCII.GetString(buffer);

            for (indice = 0; indice < buffer.Length; indice++)
            {
                buffer[indice] = byte.MinValue;
            }
            //Eliminamos la parte vacía de la cadena recibida
            if (cadena.Contains("\0"))
            {
                cadena = cadena.Substring(0, cadena.IndexOf('\0'));
            }
            //Si recibimos un /r entonces es el final del código de la instrucción IR
            if (cadena.Contains("\r"))
            {
                cadena = cadena.Substring(0, cadena.IndexOf('\r'));
                ok = true;
            }
            //Si no es el final hay que recibir el resto
            else
```

```
            {
                ok = false;
                cliente.BeginReceive(buffer, 0, 255, SocketFlags.None, new
AsyncCallback(receiveCallback), cliente);
            }

            if (cadena.StartsWith("GC-IRL"))
            {
                recibido = cadena;
            }
            else
            {
                recibido = recibido + cadena;
            }
            //Si tenemos la cadena completa pasamos a almacenarla
            if (recibido != null && ok)
            {
                //Primero hay que descodificar
                recibido = convierte_cadena(recibido);
                if (cadena_correcta)
                {
                    //Guardamos la cadena en la matriz
                    instrucciones[i] = instrucciones[i] + recibido;
                    toolStripStatusLabel1.Text = "Comando \"" + instrucciones[i].Substring(0,
instrucciones[i].IndexOf(';')) + "\" guardado. Introduzca un nuevo nombre y pulse Aceptar.";
                    i++;
                }
                else
                {
                    toolStripStatusLabel1.Text = "No se recibió el comando \"" + instrucciones[i].Substring(0,
instrucciones[i].IndexOf(';')) + "\", vuelva a enviarlo por favor";
                    cliente.BeginReceive(buffer, 0, 255, SocketFlags.None, new
AsyncCallback(receiveCallback), cliente);
                }
            }
        }
        catch (Exception)
        {}
    }
```

Como vemos en este método de devolución de llamada, la recepción del código correspondiente a la pulsación de un botón de un mando a distancia no se recibe en una única transmisión asíncrona. Lo único que sabemos es que ese código forma una cadena de caracteres que comienza con *"GC-IRL"* y termina con el carácter retorno de carro *'/r'*. Mientras no recibamos ese retorno de carro seguiremos ejecutando el método de recepción asíncrona y sumando cada trozo de cadena recibido. Una vez tenemos el código completo, tenemos que descodificarlo pues como ya sabemos el GC-IRL aplica un algoritmo de codificación a la cadena de caracteres.

```
private string convierte_cadena(string cadena)
    {
        string resultado;
        string[] codificados = new string[26];
        int indice;
```

```
int imatriz;
int jmatriz;
try
{
   cadena_correcta = true;
   for (imatriz = 0; imatriz < codificados.Length; imatriz++)
   {
      codificados[imatriz] = string.Empty;
   }
   imatriz = 0;
   jmatriz = 0;
   indice = cadena.IndexOf(',', 0);
   indice = +cadena.IndexOf(',', indice + 1) + 1;
   resultado = cadena.Substring(7, indice-7); //Elimino GC-IRL
   while(indice < cadena.Length)
   {
      if (char.IsDigit(cadena, indice))
      {
         codificados[imatriz] += cadena.Substring(indice, 1);
         jmatriz++;
         resultado += cadena[indice];
      }
      else if (cadena[indice].Equals(',') && !cadena[indice - 1].Equals(','))
      {
         if (codificados[imatriz].Contains(","))
         {
            imatriz++;
            jmatriz = 0;
         }
         else if (!codificados[imatriz].Equals(string.Empty))
         {
            codificados[imatriz] += ",";
            jmatriz++;
         }
         resultado += ",";
      }
      else if (char.IsUpper(cadena, indice))
      {
         if (!codificados[imatriz].Equals(string.Empty))
         {
            if (!codificados[imatriz].Contains(","))
            {
               codificados[imatriz] = string.Empty;
            }
            else
            {
               imatriz++;
            }
            jmatriz = 0;
         }
         if (cadena[indice] - 65 < 0)
         {
            resultado = "Carácter no válido en la cadena de entrada";
            indice = cadena.Length - 1;
            cadena_correcta = false;
         }
         else if (codificados[cadena[indice] - 65].Equals(string.Empty))
         {
            resultado = "Se codificó una letra inexistente";
            indice = cadena.Length - 1;
```

```
                cadena_correcta = false;
            }
            else
            {
                resultado += "," + codificados[cadena[indice] - 65];
            }
        }
        else
        {
            resultado = "Cadena de entrada errónea";
            indice = cadena.Length-1;
            cadena_correcta = false;
        }

        indice++;
    }
  }
  catch (Exception)
  {
    resultado = "Ocurrió un error";
    cadena_correcta = false;
  }

  return resultado;
}
```

Esta función recupera la cadena de caracteres original que recibió el GC-IRL desde el mando de control usado por el usuario. Para ello almacena cada pareja de números separados por el carácter ‘,’ en una matriz. Si en lugar de una pareja de números encuentra una letra consultará en la matriz con qué pareja de números se corresponde teniendo en cuenta que dichas parejas están ordenadas en la matriz. Dicho de otro modo, la pareja de la primera posición se corresponde con la letra A, la segunda con la letra B, y así sucesivamente. Tras finalizar el método *convierte_cadena*, guardamos el código en la posición de la matriz de cadenas de caracteres correspondiente junto al nombre asociado por el usuario y separados ambos por el carácter ‘;’.

Una vez terminada la recepción de los códigos IR, y a petición del usuario, se llamará al método *crearXML* de la clase *ArchivoXML* (clase que contiene todos los métodos de manejo de ficheros XML que usamos en la aplicación). A éste método le pasamos la matriz de cadenas de caracteres que contiene todos los códigos guardados con sus respectivos nombres asociados.

```
public static void crearXML(string[] instrucciones)
    {
      int indice;
      int tamaño;
      try
```

```
        {
            XmlDocument archivoXML = new XmlDocument();
            XmlTextWriter escritorXML = new XmlTextWriter("..\\..\\..\\Registro\\" + instrucciones[0] +
".xml", Encoding.UTF8);
            //Usa indentación por legibilidad
            escritorXML.Formatting = Formatting.Indented;
            //Escribe la declaración del XML
            escritorXML.WriteStartDocument();
            escritorXML.WriteStartElement(instrucciones[0]);

            indice = 1;
            while (indice < instrucciones.Length && !instrucciones[indice].Equals(string.Empty))
            {
                escritorXML.WriteStartElement("comando");
                escritorXML.WriteElementString("nombre", instrucciones[indice].Substring(0,
instrucciones[indice].IndexOf(';')));
                tamaño = instrucciones[indice].Substring(0, instrucciones[indice].IndexOf(';')).Length + 1;
                escritorXML.WriteElementString("codigo",
instrucciones[indice].Substring(instrucciones[indice].IndexOf(';') + 1, instrucciones[indice].Length -
tamaño));
                escritorXML.WriteEndElement();
                indice++;
            }
            escritorXML.WriteEndElement();
            escritorXML.WriteEndDocument();
            escritorXML.Flush();
            escritorXML.Close();
        }
        catch (Exception)
        {}
    }
```

El formato del archivo XML que creamos es el siguiente:

```
<nombre del archivo>
        <comando>
                <nombre>nombre del primer comando IR</nombre>
                <codigo>cadena de caracteres que representa el primer código IR</codigo>
        </comando>
        <comando>
                <nombre>nombre del segundo comando IR</nombre>
                <codigo>cadena de caracteres que representa el segundo código IR</codigo>
        </comando>
        …
</nombre del archivo>
```

Con esto tendríamos guardado los códigos en un archivo XML listo para ser usado en cualquier momento.

EnviaIR

EnviaIR.cs es el segundo de los formularios que contiene la aplicación y nos servirá para proporcionar al usuario un control remoto virtual desde su PC que haga uso del GC-100 y de los archivos XML generados con el primer formulario.

El proceso inicial es el mismo: establecer una conexión mediante Socket con el GC-100 a la IP correspondiente (esta vez el número de puerto TCP es el 4998). Una vez conectado, el usuario podrá cargar cualquiera de los fichero XML disponibles, es decir, el programa presentará por pantalla los nombres de los códigos IR almacenados en el fichero elegido. Para ello usamos el método *leerXML*, al que le pasamos el nombre del fichero elegido por el usuario.

```
public static string[] leerXML(string nombre_archivo)
    {
      string[] resultado = new string[50];
      int i;
      try
      {
        for (i = 0; i < resultado.Length; i++)
        {
          resultado[i] = string.Empty;
        }
        XmlDocument archivoXML = new XmlDocument();
        archivoXML.Load("..\\..\\..\\Registro\\" + nombre_archivo + ".xml");
        XmlNodeList raiz = archivoXML.GetElementsByTagName(nombre_archivo);
        XmlNodeList lista = ((XmlElement)raiz[0]).GetElementsByTagName("comando");
        resultado[0] = nombre_archivo;
        i = 1;
        foreach (XmlElement nodo in lista)
        {
          XmlNodeList nNombre = nodo.GetElementsByTagName("nombre");
          XmlNodeList nCodigo = nodo.GetElementsByTagName("codigo");
          resultado[i] = nNombre[0].InnerText + ";" + nCodigo[0].InnerText;
          i++;
        }
      }
      catch (Exception)
      {
        resultado[0] = "Ha ocurrido un error al intentar leer el archivo XML";
      }
      return resultado;
    }
```

Nos devuelve una matriz de cadenas de caracteres con el mismo formato que la que usábamos para crear el archivo XML, es decir, el nombre del archivo en primer lugar y, a continuación, cada uno de los comandos IR con su nombre y código separados por el carácter ‘;’. Tras esto el usuario tendrá el control virtual deseado en la pantalla de su PC, sólo tendrá que elegir un comando cada vez y el puerto o los puertos de salida de los seis que tiene el GC-100.

```
private void bt_Enviar_Click(object sender, EventArgs e)
    {
      int i;
      bool encontrado;
      bool chequeado;
```

```
            int tamaño1;
            int tamaño2;
            String comando;
            try
            {
               i=0;
               encontrado = false;
               while (i < lista_comandos.Items.Count && !encontrado)
               {
                  if (lista_comandos.GetSelected(i))
                  {
                     encontrado = true;
                  }
                  i++;
               }

               if (encontrado)
               {
                  enviado = false;
                  chequeado = false;
                  tamaño1 = instrucciones[i].Substring(0, instrucciones[i].IndexOf(',')).Length;
                  tamaño2 = tamaño1 - instrucciones[i].IndexOf(';') - 1;

                  if (cb_IR1.Checked)
                  {
                     chequeado = true;
                     comando = "sendir,4:1,1," + instrucciones[i].Substring(instrucciones[i].IndexOf(';') + 1,
tamaño2) + ",1,1" + instrucciones[i].Substring(instrucciones[i].IndexOf(','), instrucciones[i].Length -
tamaño1) + "\r";
                     byte[] bufferEnviado = Encoding.ASCII.GetBytes(comando);
                     controlRemoto.BeginSend(bufferEnviado, 0, bufferEnviado.Length, SocketFlags.None,
new AsyncCallback(sendCallback), controlRemoto);
                     System.Threading.Thread.Sleep(500);
                  }

                  if (cb_IR2.Checked)
                  {
                     chequeado = true;
                     comando = "sendir,4:2,2," + instrucciones[i].Substring(instrucciones[i].IndexOf(';') + 1,
tamaño2) + ",1,1" + instrucciones[i].Substring(instrucciones[i].IndexOf(','), instrucciones[i].Length -
tamaño1) + "\r";
                     byte[] bufferEnviado = Encoding.ASCII.GetBytes(comando);
                     controlRemoto.BeginSend(bufferEnviado, 0, bufferEnviado.Length, SocketFlags.None,
new AsyncCallback(sendCallback), controlRemoto);
                     System.Threading.Thread.Sleep(500);
                  }

                  if (cb_IR3.Checked)
                  {
                     chequeado = true;
                     comando = "sendir,4:3,3," + instrucciones[i].Substring(instrucciones[i].IndexOf(';') + 1,
tamaño2) + ",1,1" + instrucciones[i].Substring(instrucciones[i].IndexOf(','), instrucciones[i].Length -
tamaño1) + "\r";
                     byte[] bufferEnviado = Encoding.ASCII.GetBytes(comando);
                     controlRemoto.BeginSend(bufferEnviado, 0, bufferEnviado.Length, SocketFlags.None,
new AsyncCallback(sendCallback), controlRemoto);
                     System.Threading.Thread.Sleep(500);
                  }

                  if (cb_IR4.Checked)
```

```
                {
                    chequeado = true;
                    comando = "sendir,5:1,4," + instrucciones[i].Substring(instrucciones[i].IndexOf(';') + 1,
tamaño2) + ",1,1" + instrucciones[i].Substring(instrucciones[i].IndexOf(','), instrucciones[i].Length -
tamaño1) + "\r";
                    byte[] bufferEnviado = Encoding.ASCII.GetBytes(comando);
                    controlRemoto.BeginSend(bufferEnviado, 0, bufferEnviado.Length, SocketFlags.None,
new AsyncCallback(sendCallback), controlRemoto);
                    System.Threading.Thread.Sleep(500);
                }

                if (cb_IR5.Checked)
                {
                    chequeado = true;
                    comando = "sendir,5:2,5," + instrucciones[i].Substring(instrucciones[i].IndexOf(';') + 1,
tamaño2) + ",1,1" + instrucciones[i].Substring(instrucciones[i].IndexOf(','), instrucciones[i].Length -
tamaño1) + "\r";
                    byte[] bufferEnviado = Encoding.ASCII.GetBytes(comando);
                    controlRemoto.BeginSend(bufferEnviado, 0, bufferEnviado.Length, SocketFlags.None,
new AsyncCallback(sendCallback), controlRemoto);
                    System.Threading.Thread.Sleep(500);
                }

                if (cb_IR6.Checked)
                {
                    chequeado = true;
                    comando = "sendir,5:3,6," + instrucciones[i].Substring(instrucciones[i].IndexOf(';') + 1,
tamaño2) + ",1,1" + instrucciones[i].Substring(instrucciones[i].IndexOf(','), instrucciones[i].Length -
tamaño1) + "\r";
                    byte[] bufferEnviado = Encoding.ASCII.GetBytes(comando);
                    controlRemoto.BeginSend(bufferEnviado, 0, bufferEnviado.Length, SocketFlags.None,
new AsyncCallback(sendCallback), controlRemoto);
                    System.Threading.Thread.Sleep(500);
                }

                if (!chequeado)
                {
                    toolStripStatusLabel1.Text = "Debe seleccionar uno o varios puertos IR de salida";
                }
            }
            else
            {
                toolStripStatusLabel1.Text = "No ha seleccionado el comando o éste no existe";
            }
        }
        catch (Exception)
        {
            toolStripStatusLabel1.Text = "Ocurrió un error inesperado, compruebe la conexión";
            lb_DirIP.Enabled = true;
            tb_dirIP.Enabled = true;
            bt_Conectar.Enabled = true;
            bt_Desconectar.Enabled = false;
            lb_nombre_archivo.Enabled = false;
            tb_nombre_archivo.Enabled = false;
            bt_Cargar.Enabled = false;
            lb_seleccionaIR.Enabled = false;
            cb_IR1.Enabled = false;
            cb_IR2.Enabled = false;
            cb_IR3.Enabled = false;
            cb_IR4.Enabled = false;
```

```
            cb_IR5.Enabled = false;
            cb_IR6.Enabled = false;
            lb_lista_comandos.Enabled = false;
            lista_comandos.Enabled = false;
            bt_Enviar.Enabled = false;
        }
    }
```

Para llevar a cabo la transmisión de la orden IR a través del GC-100 usamos el método *BeginSend*, que envía datos asíncronamente a un objeto Socket conectado (en nuestro caso es *controlRemoto*).

controlRemoto.BeginSend(bufferEnviado, 0, bufferEnviado.Length, SocketFlags.None, new AsyncCallback(sendCallback), controlRemoto);

Donde *sendCallback* es el método de devolución de llamada que se ejecutará tras completarse la transmisión.

```
private void sendCallback(IAsyncResult ar)
    {
        Socket tmp_controlRemoto;
        int bytes_enviados;
        try
        {
            tmp_controlRemoto = (Socket)ar.AsyncState;
            bytes_enviados = tmp_controlRemoto.EndSend(ar);
            tmp_controlRemoto.BeginReceive(buffer, 0, 255, SocketFlags.None, new
AsyncCallback(receiveCallback), tmp_controlRemoto);
        }
        catch (Exception)
        {
            toolStripStatusLabel1.Text = "Se produjo un error durante el envío";
        }
    }
```

En este método ejecutado tras el envío llevaremos acabo el inicio de la recepción con el método *BeginReceive* para comprobar el mensaje de confirmación que nos debe devolver el GC-100. El método de devolución de llamada es *receiveCallback*.

```
private void receiveCallback(IAsyncResult ar)
    {

        int bytesLeidos;
        string cadena = string.Empty;
        try
        {
            bytesLeidos = controlRemoto.EndReceive(ar);
            cadena = Encoding.ASCII.GetString(buffer);

            if (cadena.StartsWith("completeir"))
            {
```

```
            toolStripStatusLabel1.Text = cadena.Substring(0,cadena.IndexOf('\r'));
            enviado = true;
        }
        else if(cadena.StartsWith("unknowncommand"))
        {
            toolStripStatusLabel1.Text = cadena.Substring(0,cadena.IndexOf('\r'));
            enviado = false;
        }
    }
    catch (Exception)
    {
        toolStripStatusLabel1.Text = "Ocurrió un error inesperado y no se completó el envío";
        enviado = false;
    }
}
```

Con este formulario hemos obtenido un control remoto a disposicón del usuario que se ejecuta desde el PC. Esto nos permite gobernar cualquier dispositivo electrónico con control remoto a través del GC-100 siempre que hayamos memorizado en el sistema los códigos correspondientes a su mando a distancia (con *RecibeIR.cs*).

ReceptorIR

El tercer formulario de este programa nos va a brindar la posibilidad de usar el GC-100 como un receptor y emisor de códigos IR al mismo tiempo. Vamos a convertir el Adaptador de Red en un puente que nos permite recibir una orden de un mando a distancia, reconocerla, y reenviarla. La idea es poder tener el dispositivo electrónico destino de la orden IR en una habitación distinta a aquella donde se encuentra el receptor (en este caso el GC-IRL).

Estableceremos dos conexiones mediante Socket con el GC-100: una al puerto 4998 para la emisión IR y otra al puerto donde esté conectado el GC-IRL. El procedimiento es similar al visto anteriormente para los dos primeros formularios. En este caso usamos dos métodos de devolución de llamada distintos para atender al método *BeginConnect*.

Tras esto el usuario introducirá el nombre del fichero XML a usar como fuente de códigos IR. Para ello volvemos a hacer uso del método *leerXML*. En este caso no se creará un control virtual sino que se tendrá el sistema listo para recibir códigos IR desde el GC-IRL que coincidan con alguno de los códigos guardados en el fichero XML seleccionado. Tras ser reconocido por el sistema éste lo reenvía al GC-100 para que sea emitido por los puertos de salida IR seleccionados previamente por el usuario.

Para llevar a cabo la comparación entre cádenas de código IR creamos el método *comparar*. Hay que tener en cuenta que se permite un margen de tolerancia de 3 entre los valores numéricos y que el sistema de codificación mayoritario es el RC6.

```
private void comparar(string recibido)
      {
         int i;
         int j;
         int m;
         int n;
         int indice1;
         int indice2;
         double valor1;
         double valor2;
         double valor1RC6;
         double valor2RC6;
         string cadena;
         string aux;
         bool encontrado;
         bool final;
         int cuenta1;
         int cuenta2;
         bool RC6;
         try
         {
            encontrado = false;
            i = 1;
            //Comparamos con cada cadena incluida en la matriz
            while (!instrucciones[i].Equals(string.Empty) && i < instrucciones.Length && !encontrado)
            {
               valor1RC6 = 0; //Nos ayudará a identificar código RC6
               valor2RC6 = 0;
               RC6 = false;
               cuenta1 = 0;
               cuenta2 = 0;
               final = false;
               aux = recibido; //Cadena recibida del GC-100
               //Elimino el nombre del comando
               indice1 = instrucciones[i].Substring(0, instrucciones[i].IndexOf(';')).Length;
               //Cadena guardada con la que haremos la comparación
               cadena = instrucciones[i].Substring(instrucciones[i].IndexOf(';') + 1, instrucciones[i].Length
- indice1 - 1);

               //Los números del 0 al 9 en ASCII van del 48 al 57
               indice1 = cadena.IndexOf(',');
               valor1 = 0;
               //Con el siguiente bucle obtengo el valor de frecuencia
               for (j = 0; j < indice1; j++)
               {
                  valor1 += (cadena[j] - 48) * Math.Pow(10, (indice1 - 1 - j));
               }
               indice2 = aux.IndexOf(',');
               valor2 = 0;
               for (j = 0; j < indice2; j++)
               {
                  valor2 += (aux[j] - 48) * Math.Pow(10, (indice2 - 1 - j));
               }
               //Doy un margen de tolerancia de 3000 Hz para la frecuencia
```

```
if ((valor2 - 3000) <= valor1 && valor1 <= (valor2 + 3000))
{
   do
   {
      m = indice1 + 1;
      indice1 = cadena.IndexOf(',', indice1 + 1);
      n = indice2 + 1;
      indice2 = aux.IndexOf(',', indice2 + 1);
      //IndexOf devuelve un -1 si no se encuentra el carácter
      if (!indice1.Equals(-1))
      {
         if (!indice2.Equals(-1))
         {
            cuenta1++;
            cuenta2++;
            valor1 = 0;
            //Hallamos el valor númerico hasta la siguiente ','
            for (j = m; j < indice1; j++)
            {
               valor1 += (cadena[j] - 48) * Math.Pow(10, (indice1 - 1 - j));
            }

            valor2 = 0;
            for (j = n; j < indice2; j++)
            {
               valor2 += (aux[j] - 48) * Math.Pow(10, (indice2 - 1 - j));
            }

            //El margen de tolerancia es 3
            if (!((valor2 - 3) <= valor1 && valor1 <= (valor2 + 3)))
            {
               //Si aún no hemos hallado la conmutación RC6
               //Y no hemos llegado al límite para encontrarla
               if (!RC6 && (cuenta1 < 15))
               {
                  //Si es el primer intento
                  if (valor1RC6.Equals(0) && valor2RC6.Equals(0))
                  {
                     //Almacenamos los valores
                     valor1RC6 = valor1;
                     valor2RC6 = valor2;
                     //cuenta2 delimitará el rango para hallar RC6
                     cuenta2 = 0;
                  }
                  else if (cuenta2.Equals(2))
                  {
                     //Comparamos los actuales con los almacenados con un margen de 3
                     if ((valor1 - 3) <= valor2RC6 && valor2RC6 <= (valor1 + 3) &&
                        (valor2 - 3) <= valor1RC6 && valor1RC6 <= (valor2 + 3))
                     {
                        RC6 = true; //Hemos hallado la conmutación RC6
                     }
                     else
                     {
                        final = true;
                     }
                  }
                  else
                  {
                     final = true;
```

```
                        }
                    }
                    else
                    {
                        final = true;
                    }
                }
            }
            else
            {
                final = true;
            }
        }
        else
        {
            final = true;
            if (indice2.Equals(-1) && cuenta2 >= 2)
            {
                encontrado = true;
            }
        }
    }
    while (!final);
}
i++;
}
if (encontrado)
{
    toolStripStatusLabel1.Text = "El sistema ha recibido el comando \"" + instrucciones[i-
1].Substring(0, instrucciones[i-1].IndexOf(';')) + "\"";
    enviar(instrucciones[i - 1]);
}
else
{
    toolStripStatusLabel1.Text = "El sistema ha recibido un comando desconocido";
}
}
catch (Exception)
{ }
}
```

Si el sistema reconoce el código entrante como uno de los que tiene almacenado podrá reproducirlo o simplemente hacerse eco de ese evento (esto será fundamental en la integración con la arquitectura UPnP).

Y con esto tenemos los tres formularios de la aplicación.

5.1.2 Integración con la arquitectura UPnP

Ya tenemos hecha una aplicación que aprovecha el Adaptador de Red GC-100 para recibir y transmitir códigos IR y así controlar todo tipo de dispositivos electrónicos. Ahora lo que nos interesa es integrar nuestro sistema en una red domótica con arquitectura UPnP.

Nuestro sistema va a aportar dos servicios a la red:

1. Controlar cualquier dispositivo electrónico usando códigos de control IR almacenados en ficheros XML.
2. Detectar una orden emitida desde un mando de control a un receptor de infrarrojos integrado en la red (en este caso, el GC-IRL).

Para cumplir con el primero de ellos pondremos a disposición de la red un método con el que la red pasará una variable de tipo string con el nombre de la orden IR (de nombre *Orden),* otra variable de tipo string con el nombre del fichero XML donde se encuentra esa orden (de nombre *Dispositivo*), y una variable de tipo byte con el número del puerto de salida IR del GC-100 que desea utilizar (de nombre *Puerto*).

```
private void enviar_upnp(System.String orden, System.String dispositivo, System.Byte puertoIR)
    {
      int tamaño1;
      int tamaño2;
      string aux;
      string[] archivo = new string[50];
      bool encontrado;
      int i;
      string salidaIR;

      try
      {
        archivo = ArchivoXML.leerXML(dispositivo);
        if (!archivo[0].Equals("Ha ocurrido un error al intentar leer el archivo XML"))
        {
          encontrado = false;
          i = 1;
          while (!archivo[i].Equals(string.Empty) && i < archivo.Length && !encontrado)
          {
            if (orden.Equals(archivo[i].Substring(0, archivo[i].IndexOf(';'))))
            {
              encontrado = true;
            }
            else
            {
              i++;
            }
          }

          if (encontrado)
          {
            salidaIR = string.Empty;
            if (puertoIR.Equals(1))
            {
              salidaIR = "4:1";
            }
            else if (puertoIR.Equals(2))
            {
              salidaIR = "4:2";
```

```
                }
                else if (puertoIR.Equals(3))
                {
                   salidaIR = "4:3";
                }
                else if (puertoIR.Equals(4))
                {
                   salidaIR = "5:1";
                }
                else if (puertoIR.Equals(5))
                {
                   salidaIR = "5:2";
                }
                else if (puertoIR.Equals(6))
                {
                   salidaIR = "5:3";
                }

                tamaño1 = archivo[i].Substring(0, archivo[i].IndexOf(',')).Length;
                tamaño2 = tamaño1 - archivo[i].IndexOf(';') - 1;
                aux = "sendir," + salidaIR + ",1," + archivo[i].Substring(archivo[i].IndexOf(';') + 1,
tamaño2) + ",1,1" + archivo[i].Substring(archivo[i].IndexOf(','), archivo[i].Length - tamaño1) + "\r";

                byte[] bufferEnviado = Encoding.ASCII.GetBytes(aux);
                envio.BeginSend(bufferEnviado, 0, bufferEnviado.Length, SocketFlags.None, new
AsyncCallback(sendCallback), envio);
             }
             else
             {
                toolStripStatusLabel1.Text = "Orden no encontrada";
             }
          }
          else
          {
             toolStripStatusLabel1.Text = "El archivo especificado no existe o hubo un error al abrirlo";
          }
       }
       catch { }
    }
```

De este modo, y aprovechando la versatilidad de la arquitectura UPnP la red puede enviar códigos IR a través del GC-100.

Para que la red sea avisada de la recepción de un código por parte del GC-IRL tendremos que añadir la llamada al método correspondiente dentro de nuestra función que compara la cadena de caracteres del código IR recibido con las que están guardadas en el sistemas. Para ellos añadimos en el programa lo siguiente:

```
if (encontrado)
          {
             toolStripStatusLabel1.Text = "El sistema ha recibido el comando \"" + instrucciones[i-
1].Substring(0, instrucciones[i-1].IndexOf(';')) + "\"";
             device.ServicioGC100.Evented_Orden = instrucciones[i - 1].Substring(0, instrucciones[i -
1].IndexOf(';'));
          }
```

De este modo la red queda avisada de la recepción de un determinado código IR y podrá actuar en consecuencia.

5.2 Manual de usuario

Vamos a redactar un pequeño manual de usario para la utilización de nuestra aplicación. En primer lugar hay que tener claro que necesitamos conocer la dirección IP del GC-100 para poder iniciar cualquier procedimiento. En cualquier caso, y en aras de facilitar el trabajo al usuario, la aplicación cuenta con un registro que almacena los últimos valores de dirección IP y puerto introducidos. Dichos valores quedan almacenados en un fichero XML que el sistema consulta cada vez que se inicia y los coloca por defecto para que el usuario no tenga que escribirlos cada vez.

La pantalla de bienvenida de la aplicación nos muestra el acceso a los tres formularios explicados previamente.

Figura 5.1 Formulario inicial de la aplicación

5.2.1 Crear fichero de instrucciones

Como sabemos tenemos tres formularios que nos aportan tres servicios distintos. Si hacemos clic en el primero de ellos veremos lo siguiente:

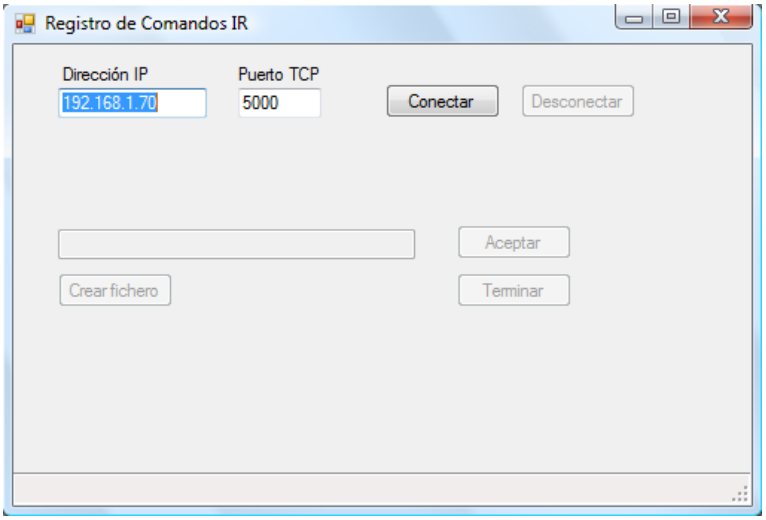

Figura 5.2 Formulario "Crear fichero de instrucciones"

Donde, como vemos, aparecen los valores de IP y puerto introducidos por última vez. Una vez que tenemos los valores correctos hacemos clic en *Conectar*.

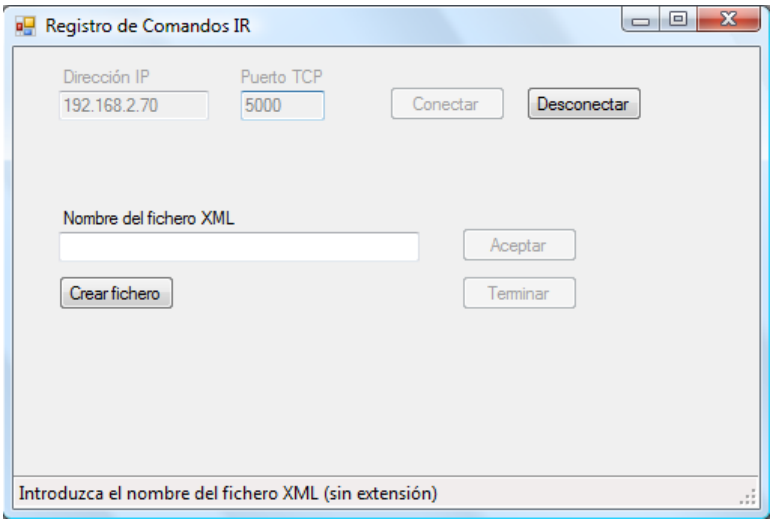

Figura 5.3 Esperando que el usuario introduzca el fichero XML

Ahora hay que escribir un nombre para el fichero XML que guardará los códigos IR que introduzcamos (por ejemplo, "Televisor"). A continuación, hacemos clic en *Crear*.

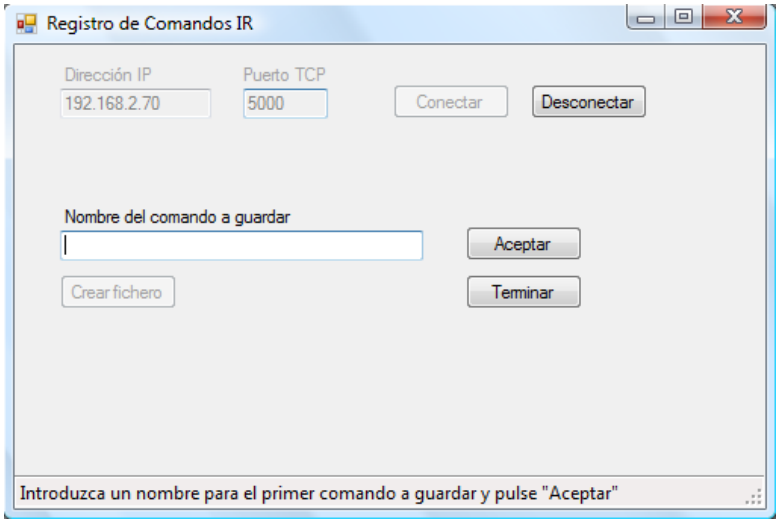

Figura 5.4 Esperando el nombre de la siguiente intrucción IR

Como se nos indica en la barra de estado, lo siguiente es escribir el nombre con el que reconocer la primera instrucción que queremos guardar (por ejemplo, "Encender") y hacer clic en *Aceptar*.

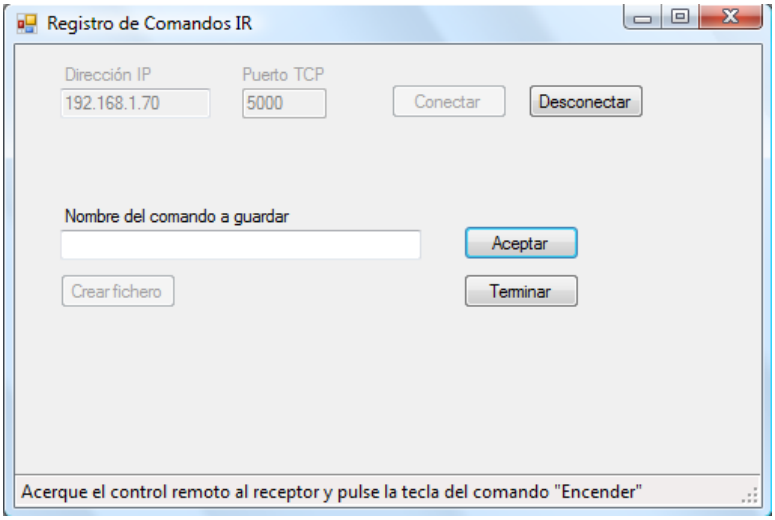

Figura 5.5 Esperando a recibir una señal IR desde el GC-IRL

Ahora es cuando tendremos que acercar el mando a distancia al receptor, es decir, el GC-IRL, a una distancia de entre 3 y 6 cm a ser posible, y pulsar la tecla de la instrucción que queremos almacenar con el nombre dado (en nuestro caso, "Encender").

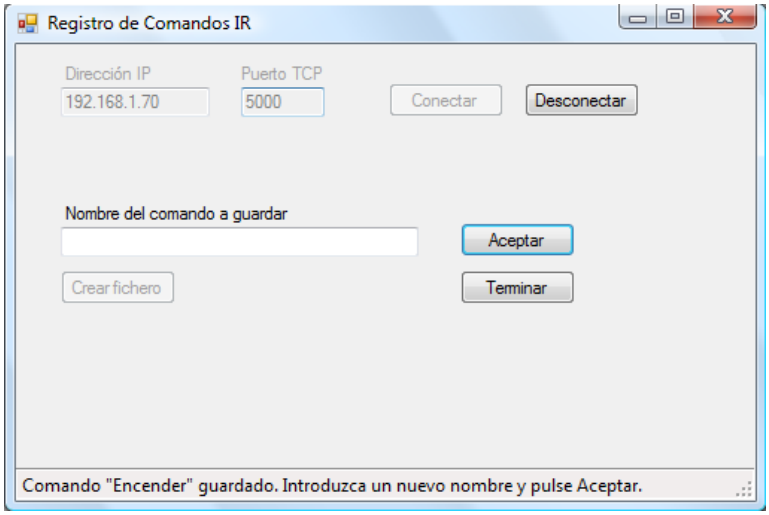

Figura 5.6 A la espera de una nueva instrucción IR

Repetiremos esta operación para todas las instrucciones que queramos almacenar. Cuando finalicemos haremos clic en "Terminar".

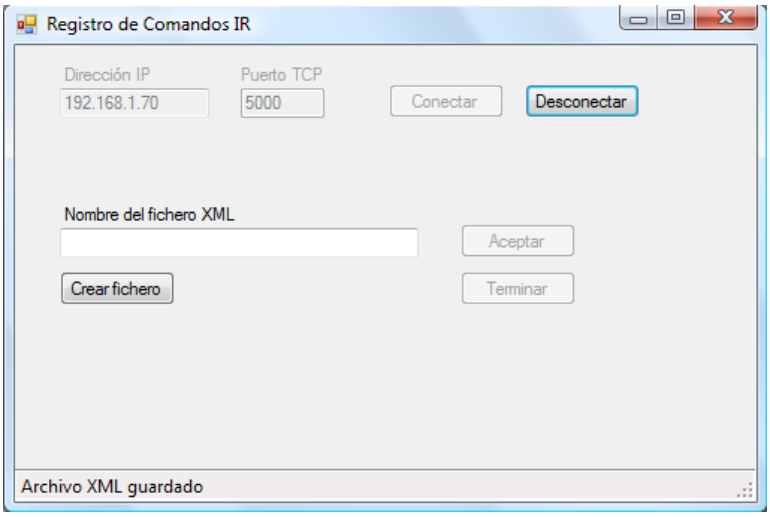

Figura 5.7 Esperando para crear un nuevo fichero XML

Ya tenemos el fichero XML con las instrucciones almacenadas a nuestra disposición.

5.2.2 Implemetar controlador IR

El segundo formulario es el que nos crea un control remoto virtual a partir de un fichero XML con las instrucciones guardadas.

Los pasos iniciales son los mismos que antes salvo que en este caso no se solicita al usuario un número de puerto TCP, ya que éste es el 4998 y la aplicación lo usa por defecto.

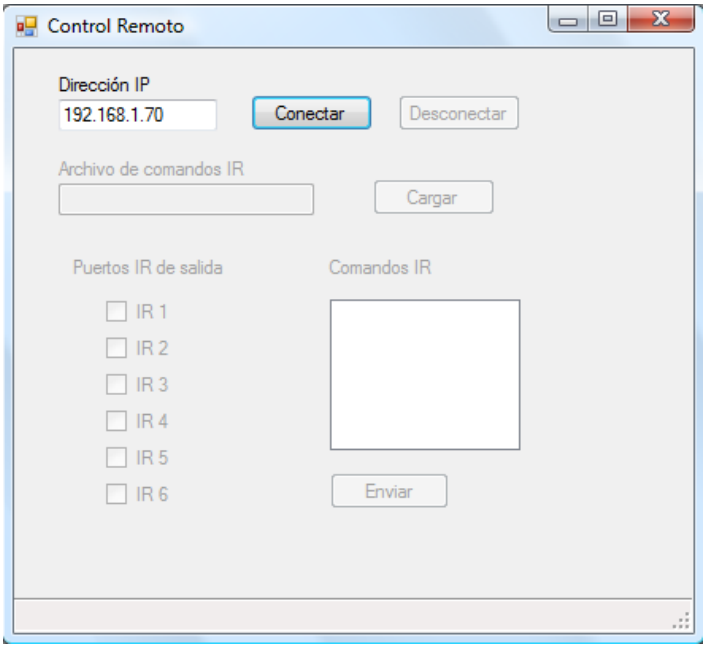

Figura 5.8 Formulario "Implementar controlador IR"

Una vez que estamos conectados al GC-100, introducimos el nombre del fichero XML del que queremos usar sus instrucciones guardadas y hacemos clic en *Cargar*.

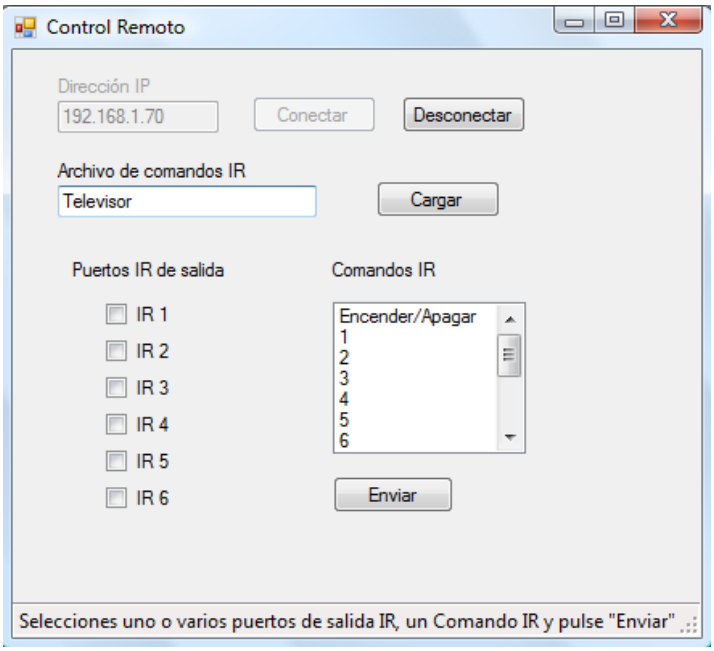

Figura 5.9 Control virtual listo para usarse

Como se nos dice en la barra de estado, ahora ya podremos enviar órdenes IR seleccionando los puertos de salida del GC-100 por los que queremos que salgan dichas órdenes. Obviamente los puertos de salida IR deben tener un emisor de infrarrojos conectado (de los que suministra Global Caché) apuntando hacia el aparato que queremos controlar. Haciendo clic en *Enviar* la instrucción será enviada al GC-100 y de éste al aparato en cuestión.

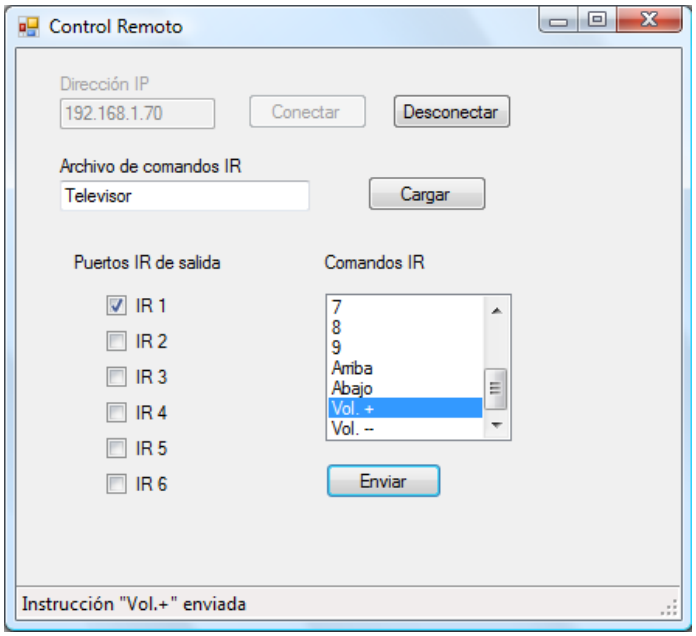

Figura 5.10 Instrucción IR enviada

5.2.3 Modo Receptor

Con el tercer formulario convertimos el sistema en un receptor que espera recibir una instrucción conocida desde el GC-IRL para reproducirla por uno de los puertos de salida IR del GC-100.

La conexión es idéntica a los casos anteriores (en este caso sí elegimos el número de puerto según sea el puerto serie del GC-100 al que el GC-IRL esté conectado).

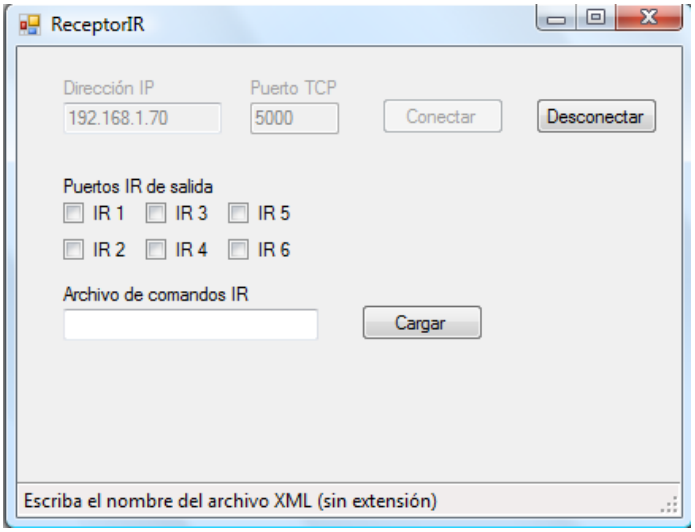

Figura 5.11 Formulario "Modo Receptor"

Como antes, introducimos el nombre del fichero XML donde estén las instrucciones IR con las que queremos comparar las que recibamos desde el GC-IRL, y señalamos el puerto o los puertos de salida IR del GC-100 por los que queramos hacer las retransmisiones.

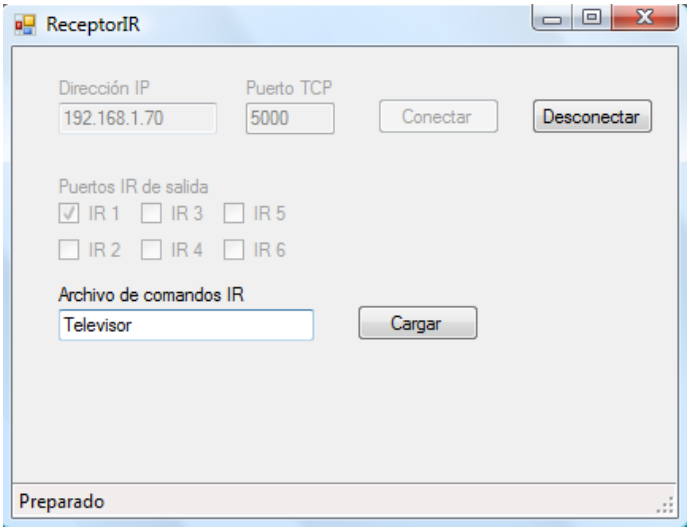

Figura 5.12 Listo para reconocer y retransmitir instrucciones IR

5.2.4 Modo Receptor integrado con UPnP

El funcionamiento es similar al visto en el apartado anterior. Una vez que esté conectado, la red podrá enviar las instrucciones IR que considere oportuno de entre todas las que contienen los ficheros XML guardados para tal fin. Además, si el sistema

recibe una instrucción IR desde el GC-IRL y ésta se encuentra en el fichero XML cargado se le notificará a la red la ocurrencia de tal evento.

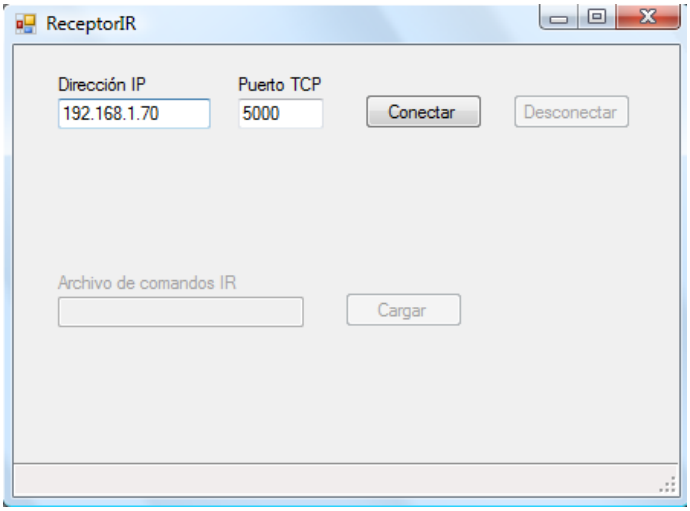

Figura 5.13 Receptor integrado con UPnP