Arduino YUN y el Internet de las cosas (IoT)

JOSÉ M. CÁMARA NEBREDA

(checam@ubu.es)

2017

CONTENIDO

INTRODUCCIÓN

El concepto de Internet de las Cosas, conocido también por su acrónimo en inglés IoT (Internet of Things), se ha extendido en los últimos tiempos a multitud de ámbitos. El origen de esta expansión se ha de encontrar en el hecho de que la conectividad es una funcionalidad cada vez más asequible para cualquier dispositivo. Este inconcreto calificativo: asequible, se puede concretar en tres aspectos básicos para la proliferación de cualquier dispositivo electrónico en el mercado:

- Coste
- Consumo
- Tamaño

De esta forma cada vez es más viable que pequeños y sencillos aparatos incorporen la capacidad de intercambiar información con el exterior sin que eso suponga que su tamaño, su precio o su consumo energético se incrementen significativamente.

La forma de alcanzar esta capacidad se puede alcanzar mediante múltiples soluciones tecnológicas. En este documento se va a explicar cómo lograrlo a través de una tecnología de gran expansión en el mercado especialmente a nivel didáctico y experimental, como es Arduino. Dentro de la creciente gama de productos relacionados, el dispositivo que nos ha parecido más adecuado para iniciarnos en ello es el modelo Arduino Yun. Este dispositivo nos proporciona varias opciones de conectividad, a saber: comunicación serie, Ethernet y Wifi. Con todo ello podemos construir aplicaciones realmente interesantes en este ámbito.

Si la motivación para adentrarse en este documento es la de construir una interfaz desde un PC a Arduino Yun es especialmente recomendable comenzar por el documento que precede a éste en el que se explica detalladamente cómo lograrlo mediante el desarrollo de aplicaciones en VisualC++. Incluso si no es éste el objetivo, el citado documento puede resultar de interés.

ARDUINO YUN

Arduino YUN es un notable miembro de la familia Arduino. Para entender rápidamente su arquitectura, podemos asumir de que se trata de un Arduino Leonardo que a su vez es similar al Arduino Micro que ya hemos utilizado (mismo procesador que Arduino Micro, pero formato de Arduino UNO), unido a un minicomputador con su propia CPU (Atheros AR9331) trabajando bajo un sistema operativo Linux (OpenWrt-Yun), espacio de almacenamiento (micro SD y USB) y conectividad (Ehternet, Wifi, host USB).

Es importante señalar que YUN se alimenta exclusivamente a 5V. No admite por lo tanto el rango habitual de la familia Arduino de 7 a 12V.

Por otra parte, es necesario tener en cuenta también que, al igual que Arduino Micro, las comunicaciones con el PC a través del puerto serie requieren de la activación de DTREnable en la configuración de propiedades del puerto en el programa PC.

El trabajo presentado en el documento anterior realizado con la shield de Arduino UNO es totalmente aprovechable para YUN. Con la salvedad de la habilitación de DTR para la comunicación serie, tanto al código en VisualC++ como el de Arduino funcionan perfectamente sin ninguna modificación.

En cualquier caso, no merece la pena adquirir un dispositivo tan complejo como YUN para hacer lo mismo que con otros Arduino. Ahora disponemos de un ordenador paralelo, conectado a través del puerto serie 1 (serial1), que nos proporciona múltiples posibilidades. De entrada puede aparecer como un reto abrumador, si uno está aprendiendo a trabajar con Arduino, a generar aplicaciones sencillas en VisualC++, adentrarse en un nuevo sistema operativo como Linux y en un entorno de programación como Pyton que es el que por defecto incorpora el entorno del procesador Atheros. Sin embargo, no es necesario tan alto esfuerzo de aprendizaje para aprovechar las principales ventajas de YUN, léase, almacenamiento y conectividad. Vamos a ir viendo cómo sacarle rendimiento poco a poco.

CONEXIÓN WIFI

Arduino YUN puede conectarse a una red inalámbrica (wifi) existente o puede crear su propia red. En el momento del arranque, el dispositivo se va a convertir en punto de acceso a su propia red. Si desplegamos el explorador de redes de nuestro ordenador o dispositivo móvil vamos a ver una nueva conexión como la que aparece en la Figura 1.

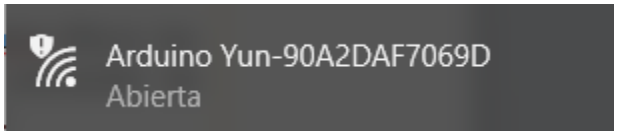

FIGURA 1

Podemos conectarnos a ella para obtener una IP y así podremos ingresar en la página de bienvenida de YUN que vemos en la Figura 2, a través de http://arduino.local o la dirección IP: 192.168.240.1.

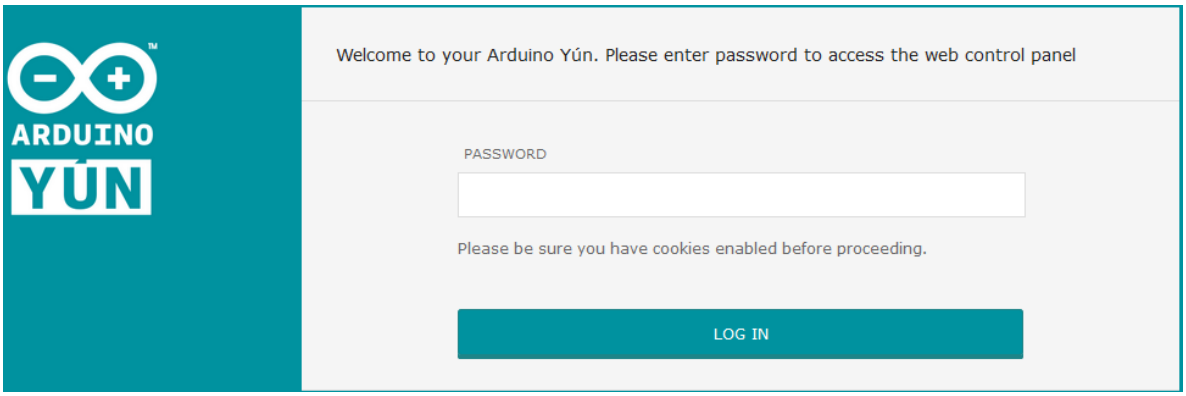

FIGURA 2

La contraseña por defecto es “arduino”. Una vez introducida veremos lo que se muestra en la Figura 3.

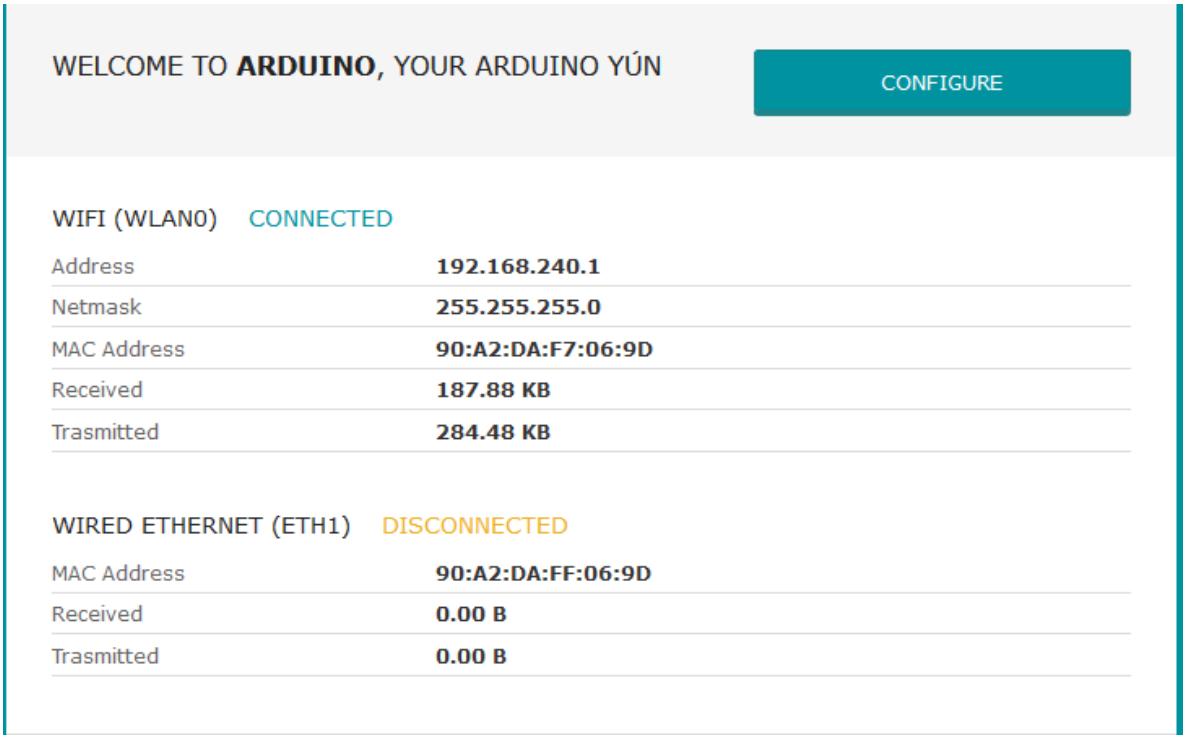

FIGURA 3

Si entramos en configuración podremos incorporar YUN a nuestra Wifi o mantenerlo como punto de acceso a su propia wifi, para lo cual liberaremos el tick en “CONFIGURE A WIRELESS NETWORK”, como se aprecia en la Figura 4. Nos quedamos con esta opción por el momento; de esta forma queda claro que para acceder a la funcionalidad de Arduino YUN vía wifi, no es necesaria ninguna infraestructura adicional.

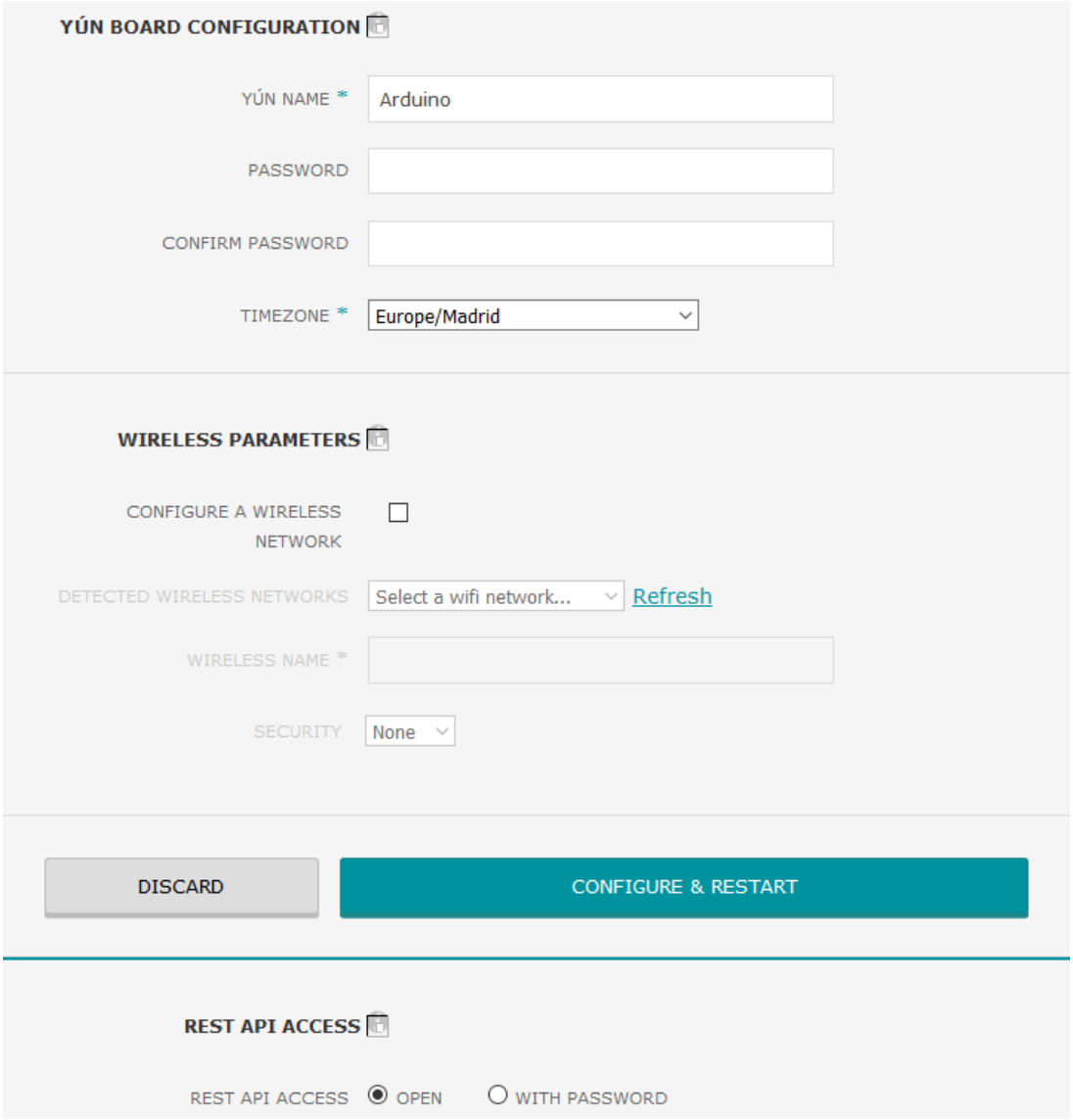

FIGURA 4

Siempre se puede recuperar la configuración inicial Wifi pulsando el botón de Reset de la misma (junto al conector USB). Se ha de mantener pulsado durante más de 5 segundos y menos de 30. Puede ser necesario desconectar y reconectar el dispositivo posteriormente.

En la Figura 4 vemos que hemos decidido (en la parte inferior) dejar el acceso abierto a la API REST. Eso nos va a permitir acceder a la periferia de Arduino desde http. A continuación pulsamos "CONFIGURE & RESTART". Podremos volver a conectarnos a Arduino en unos instantes. Como no hemos cambiado la configuración de red, podremos hacerlo de nuevo a través de http://arduino.local.

El acceso a la periferia vía REST es muy sencillo, ya que cada orden/consulta se presenta como una URL específica:

URL	Acción	Respuesta en el

		navegador
http://arduino.local/arduino/digital/13	Lee la entrada digital 13	`Pin D13 set to 1/0`
http://arduino.local/arduino/digital/13/1	Pone la entrada digital 13 a 1	`Pin D13 set to 1`
http://arduino.local/arduino/digital/13/0	Pone la entrada digital 13 a 0	`Pin D13 set to 0`

TABLA I

Para que esto sea posible es necesario que Arduino esté preparado para aceptar estas órdenes/consultas. Para ello es necesario incluir una serie de elementos en el sketch. La documentación de Arduino proporciona una buena guía llevarlo a cabo. Aquí lo vamos a simplificar para centrarnos en la lectura /escritura de señales digitales.

En primer lugar debemos incluir 3 ficheros de cabecera:

```
#include <Bridge.h> : comunicación entre los dos procesadores de la placa YUN
#include <YunServer.h>: funcionalidad de servidor web
#include <YunClient.h>: funcionalidad de cliente web
```

A continuación (en la propia cabecera del sketch) inicializamos el servidor:

```
YunServer server;
```

En setup() tendremos que iniciar la comunicación entre los procesadores y ordenar la escucha de conexiones procedentes del host local:

```
Bridge.begin();
server.listenOnLocalhost();
server.begin();
```

El bucle principal requiere de una estructura muy simple:

```
YunClient client = server.accept(); //Se crea una instancia del cliente para gestionar la
conexión.
 if (client) { //Si hay conexión de cliente
   process(client); //se procesa la petición en una función específica
   client.stop(); //se cierra la conexión una vez finalizada
 }
 delay(50); //Retardo que representa otras operaciones en un caso real
```

Las conexiones no requieren de mayor tratamiento. Lo restante es interpretar las peticiones correctamente. La función de procesamiento va a interpretar en primer lugar el tipo de comando que se recibe:

```
void process(YunClient client) {
 String command = client.readStringUntil('/');

 if (command == "digital") {
   digitalCommand(client);
 }
 if (command == "analog") {
   analogCommand(client);
 }
```

```
  if (command == "mode") {
    modeCommand(client);
  }
}
```

Los comandos que hemos presentado en la Tabla I devuelven en todos los casos la palabra "digital", seguida de "/", el número de entrada/salida digital y en caso de orden de escritura, de nuevo "/" y su estado. La función process() se queda con la palabra "digital" y llama a la función digitalCommand() para que interprete el contenido del comando. Esta función "extrae" la información restante y ejecuta la orden:

```
void digitalCommand(YunClient client) {
  int pin, value;
  pin = client.parseInt(); //Extrae el número de señal digital
  if (client.read() == '/') { //Una barra a continuación implica escritura
    value = client.parseInt();
    digitalWrite(pin, value);
  }
  else { //Si no hay barra la operación es de lectura
    value = digitalRead(pin);
  }
  client.print(F("Pin D")); //Se envía el estado de la señal al cliente
  client.print(pin);
  client.print(F(" set to "));
  client.println(value);
  String key = "D";
  key += pin;
  Bridge.put(key, String(value)); //Envío del par key/value para almacenar en Linux
}
```

PROGRAMACIÓN EN VISUAL C++

Vamos a generar un nuevo proyecto en Visual C++ - Windows Forms para comunicarnos con Arduino Yun. Para hacerlo de una manera más visual y más real, vamos a utilizar la shield "Arduino Basic I/O" de Microsystems Engineering que vemos en la Figura 5.

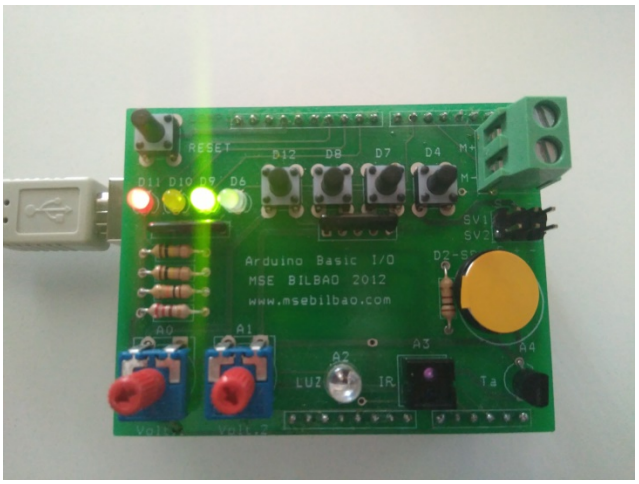

FIGURA 5

Este dispositivo nos va a permitir visualizar la ejecución de las órdenes que enviemos a Arduino, así como simular la llegada de información relevante que queramos monitorizar.

Empezaremos el proceso creando un nuevo proyecto de Windows Forms en Visual Studio como se aprecia en la Figura 6.

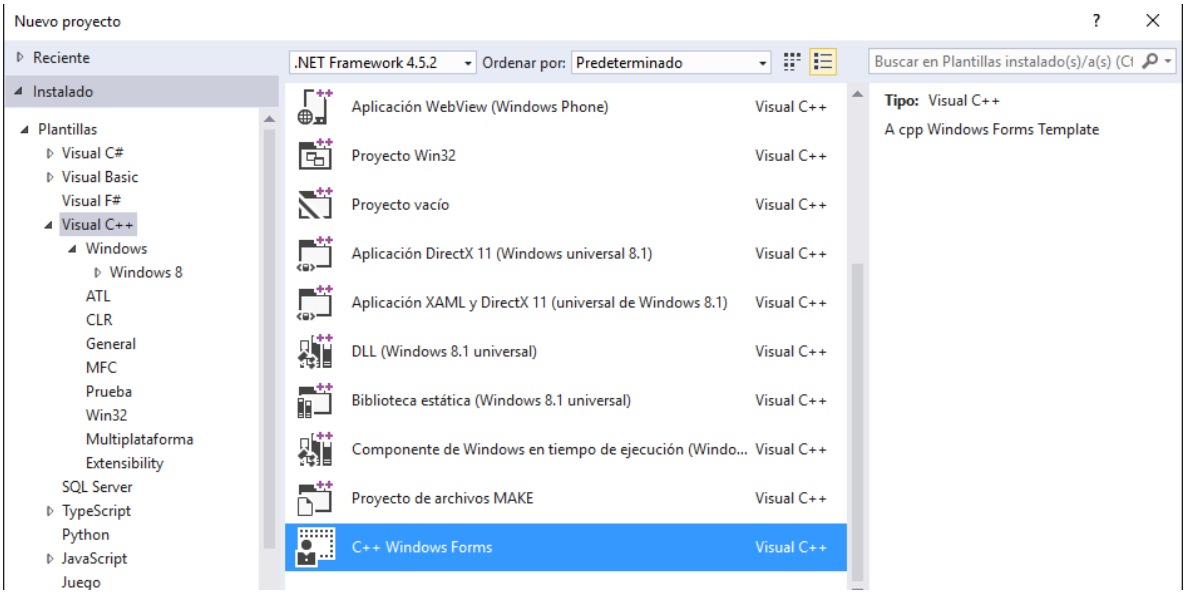

FIGURA 6

La estructura básica del proyecto incluye una ventana, denominada MyForm, a la que se asocian dos ficheros: MyForm.h (fichero de cabecera) y MyForm.cpp (código C++ que incluye el método main). Vamos a modificar el nombre de esta ventana para que resulte algo más significativo. Pulsando en la imagen de la ventana, podremos ver sus propiedades en la esquina inferior derecha. Buscaremos la propiedad “Text”, como se muestra en la Figura 7 y modificaremos su contenido por “Interfaz” por ejemplo. Veremos que inmediatamente se actualiza el texto que aparece como título de la ventana.

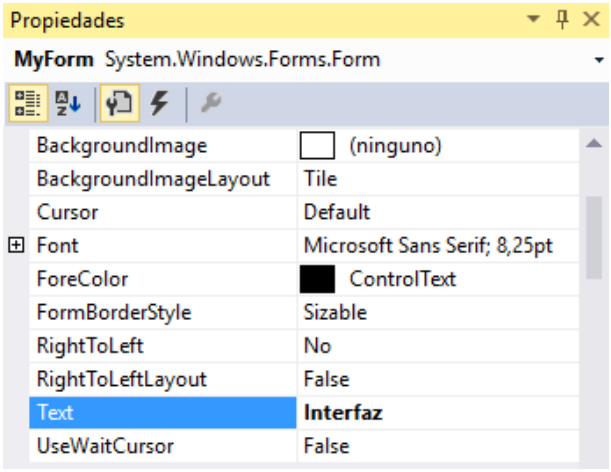

FIGURA 7

A continuación vamos a añadir una segunda ventana, con el objetivo de que se ejecute al iniciar la aplicación y que nos pida la dirección web base de Arduino. Para ello tenemos que incorporarla primero al Proyecto: Proyecto-> Agregar nuevo elemento. Seleccionaremos el nuevo formulario como se ve en la Figura 8.

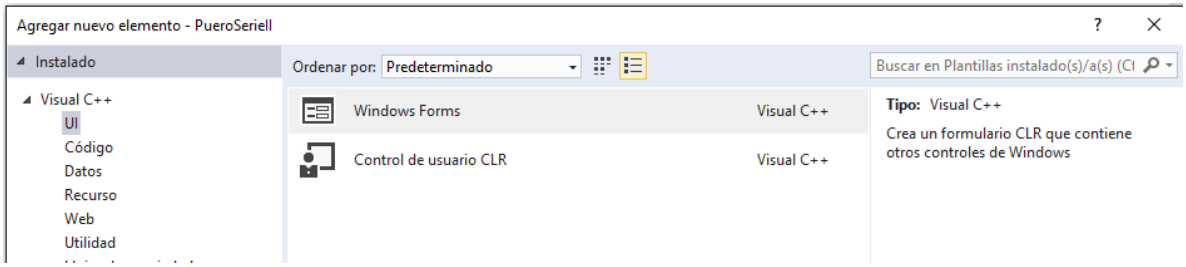

FIGURA 8

Le vamos a llamar BaseAddress y lo incluiremos el nuevo fichero de cabecera en MyForm.h:

```
#include "BaseAddress.h"
```

Podemos llamar a esta nueva ventana al inicio de la ventana principal, incluyendo en el constructor MyForm(void):

```
BaseAddress^ f = gcnew BaseAddress;
f->ShowDialog(this);
```

Esto hará que se abra la ventana BaseAddress al iniciar la apliación y que el control no vuelva a la Interfaz MyForm hasta que no se cierre esta ventana. Lo que pretendemos es que no se comience a trabajar con la interfaz si no se ha introducido la dirección de Arduino. Para que esto se lleve a cabo, tendremos que incluir un cuadro de texto (“TextBox”) en el que el usuario introduzca la URL. Le dotaremos también de una etiqueta y un botón de salida.

En el cuadro de herramientas podemos encontrar estos elementos. Para utilizarlos será necesario arrastrarlos al interior de la ventana que nos aparece en la pestaña “BaseAddress.h [Diseño]”. Veremos la imagen de la Figura 9 una vez añadidos los controles:

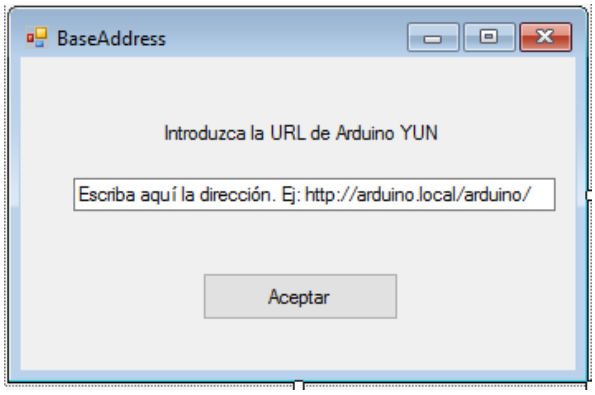

FIGURA 9

Es una interfaz sencilla, pero permite hacer lo que pretendemos. Resta codificarla no obstante. La etiqueta, cuya propiedad "Text" hemos modificado para que refleje el mensaje que queremos, no va a llevar asociada información alguna. El cuadro de texto, cuya propiedad "Text" también hemos modificado incluyendo un ejemplo de dirección URL válida, tendrá que ser leído, almacenado y comprobado. Esta va a ser la tarea más compleja. El botón de aceptar lanzará las acciones anteriores y, si la dirección es correcta, cerrará la ventana para continuar con la ejecución en la Interfaz. De lo contrario habrá que indicarle al usuario que no se encuentra a Arduino en la dirección consignada por él.

Un doble click en el botón aceptar, nos enviará a la ventana de código de BaseAddress.h en la que creará automáticamente el método que da respuesta al evento de pulsar el botón:

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
}
```

En él introduciremos el código correspondiente a las acciones anteriores. Vamos a avanzar paso a paso en esta tarea. En primer lugar vamos a ver cómo trabajar con el cliente web.

CLIENTE HTTP EN VISUAL STUDIO

La clase necesaria para conectarse a servicios web no está disponible por defecto en Visual Studio. Para instalarla tendremos que ir al explorador de soluciones y habilitar una nueva referencia (Figura 10), concretamente System.Net.Http (Figura 11).

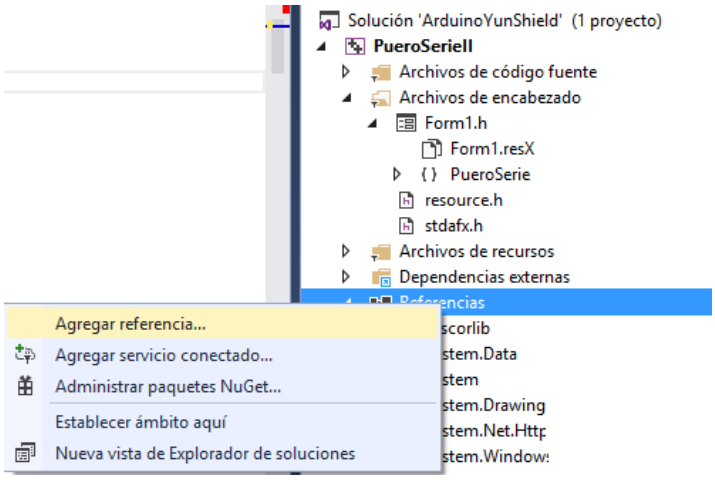

FIGURA 10

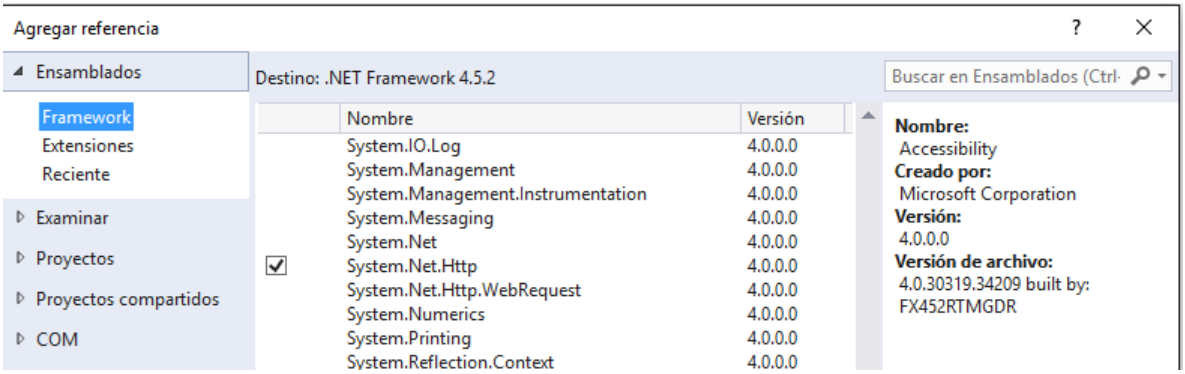

FIGURA 11

Una vez añadida, podemos habilitar los dos nuevos "namespaces" que necesitamos para acceder a las clases relacionadas con la comunicación web:

```
using namespace System::Net::Http;
using namespace System::Threading::Tasks;
```

A continuación declararemos un cliente Http como objeto privado:

```
private: HttpClient^ httpCliente;
```

y lo inicializaremos en el constructor BaseAddress(void):

```
httpCliente = gcnew HttpClient();
```

Para empezar a trabajar vamos a incluir en nuestra aplicación una nueva caja de texto, a la que llamaremos Url. El texto por defecto lo inicializaremos como: "Introduzca la URL de Arduino". Esta dirección la usaremos como dirección base para generar las órdenes y consultas subsiguientes. En nuestro caso será: http://arduino.local/arduino/. Cuando introduzcamos la Url, se convertirá en la dirección base del cliente Http.

```
private: System::Void Url_TextChanged(System::Object^  sender, System::EventArgs^
e) {
```

```
        this->httpCliente->BaseAddress = gcnew Uri(Url->Text);
}
```

Sin embargo, este método no acaba de ser seguro. El problema es que la dirección base se puede establecer una sola vez y lógicamente debe ser antes de hacer la primera consulta. Si lo hacemos depender de una caja de texto, el usuario podría obviar ese orden y lanzarse la excepción correspondiente, ya sea por no haber establecido la dirección antes de la primera consulta o por intentar modificarla después.

Una opción es prescindir de la dirección base e introducir la dirección completa en cada consulta, pero se trata de una solución muy limitada ya que si cambia el nombre que le damos a nuestro nodo Arduino, la aplicación se vuelve inservible. La solución ideal pasa por forzar al usuario a introducir la dirección base antes de acceder al resto de funcionalidad de la aplicación. Para ello lo adecuado es crear un nuevo formulario con esta misión como es el caso de nuestra ventana "BaseAddress".

El cliente http permite realizar diversas operaciones, lleva asociados numerosos métodos y propiedades y resulta de completa utilización. Vamos a simplificarlo al máximo para poder avanzar con la menor complicación posible. Nos vamos a quedar con el método "GetAsync(Url)", ya que nos va a permitir realizar las operaciones que vamos a necesitar. En principio se trata de un método para recuperar información desde el servidor web. A pesar de ello el mismo método nos va a permitir enviar órdenes, por lo que va a ser suficiente para gobernar las entradas y salidas de nuestro Arduino. Veamos un ejemplo:

```
Task<HttpResponseMessage^>^ responseTask = httpCliente->GetAsync("digital/13/0");
```

Esta línea de código genera una consulta a la dirección de base que hemos visto ya, seguida de la cadena "digital/13/0". En una sola línea implementamos una doble función: escribir un cero en la salida digital 13 y leer su estado en el objeto "responseTask". Si solo queremos leer el estado de la salida, podríamos omitir "/0". En cualquier caso vemos que con la misma función podemos leer y escribir por lo que no necesitamos recurrir a las funciones de escritura "Put" y "Post". Estas funciones nos van a pedir la Url de destino y el contenido del mensaje, cosa que en nuestro caso es lo mismo.

Tras ejecutar esta línea de código veremos que la salida digital 13, conectada al diodo integrado en la placa de YUN, se pone a cero (el diodo se apaga): la orden está ejecutada. El funcionamiento de esta instrucción no es tan sencillo no obstante. El propio nombre del método "GetAsync" introduce un matiz relevante: se trata de un método asíncrono, es decir, va a ser ejecutado por una tarea en un hilo diferente. Esto implica que la ejecución va a continuar sin que necesariamente se haya completado la acción. También implica que la información de retorno no está directamente disponible para el hilo que invocó al método, dicho de otra forma, "respnseTask" no es directamente accesible.

GESTIÓN DE LA VENTANA DE CONFIGURACIÓN

El uso de una ventana específica para configurar ciertos aspectos de funcionamiento del programa es una solución habitual. Para que funcione de forma correcta es necesario proporcionar una serie de servicios básicos.

- Generación de métodos públicos que permitan obtener los valores de configuración desde otras ventanas.
- Almacenamiento de los ajustes establecidos de manera que no sea necesario cargarlos cada vez que se ejecute la configuración.

La primera cuestión se puede abordar fácilmente creando los habituales métodos "get()" que permiten que otras clases (ventanas) obtengan los valores que se generan en la nuestra. En este caso crearemos en la zona pública de la ventana BaseAddress el método get():

```
Uri^ get(){
       return httpCliente->BaseAddress;
}
```

Por otro lado, en el constructor MyForm (void) introduciremos el siguiente código:

```
BaseAddress^ f = gcnew BaseAddress;
httpArduino = gcnew HttpClient();
f->ShowDialog(this);
httpArduino->BaseAddress = f->get();
httpArduino->GetAsync("digital/13/1");
Task<HttpResponseMessage^>^ responseTask = httpArduino->GetAsync("digital/13/0");
HttpResponseMessage^ response = responseTask->Result;
response->EnsureSuccessStatusCode();
```

Esta secuencia nos permite obtener la dirección base introducida. Adicionalmente se ha introducido una secuencia que nos permite verificar el funcionamiento correcto mediante la activación del diodo led de la salida 13 de Arduino, empleando para ello la dirección base recuperada. Esta parte se puede obviar.

La segunda cuestión pasa por la creación de un fichero de ajustes que permita almacenar valores de manera que se encuentren disponibles la próxima vez que se inicie la aplicación. Esto lo podemos hacer a través del menú Proyecto-> Agregar nuevo elemento. Lo vemos en la Figura 12.

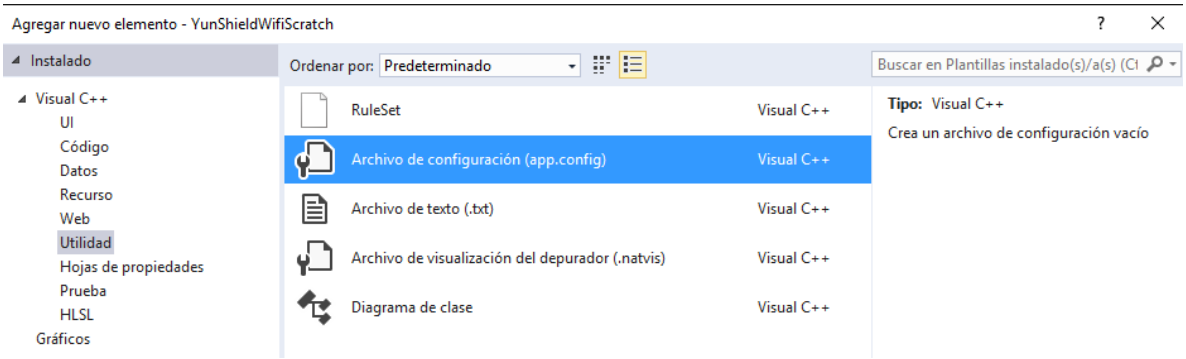

FIGURA 12

No es necesario darle nombre.

Este archivo aparecerá dentro del explorador de soluciones pero, para poder ser empleado en tiempo de ejecución, ya sea en modo “debug” o en modo “release” debe ser copiado a la carpeta en la que se encontrará el ejecutable y adquirir el siguiente nombre:

Nombreaplicación.exe.config

Esto se puede hacer a mano, pero no es necesario, A través de las propiedades del proyecto: Proyecto->Propiedades podemos configurarlo para que suceda de forma automática tras la compilación. Lo vemos en la Figura 13.

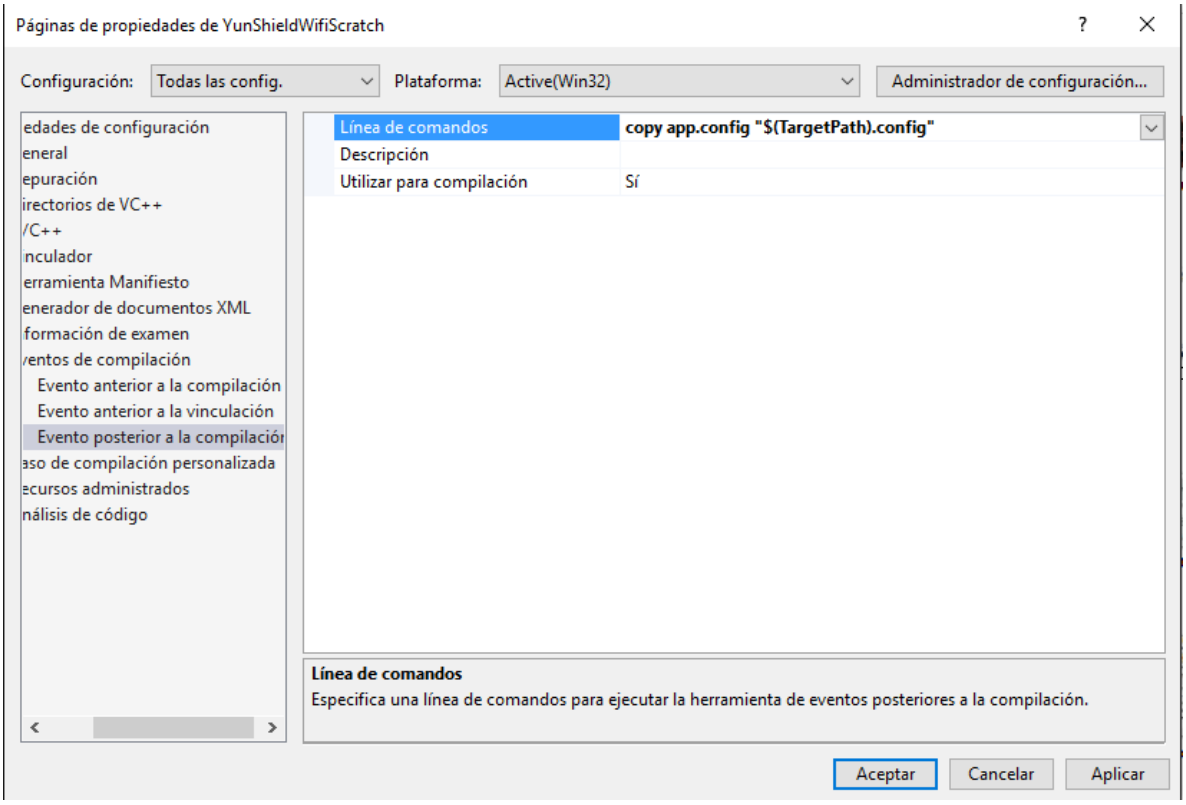

FIGURA 13

Dentro de “eventos de compilación” seleccionamos “Evento posterior a la compilación” y en “Línea de comandos” introducimos la sentencia de copiar el fichero de configuración.

Finalmente deberemos agregar una nueva referencia al proyecto. Nos situamos para ello sobre él en el explorador de soluciones y, mediante el botón derecho del ratón accedemos a “Agregar referencia”. Debemos añadir “System.Configuration” como vemos en la Figura 14.

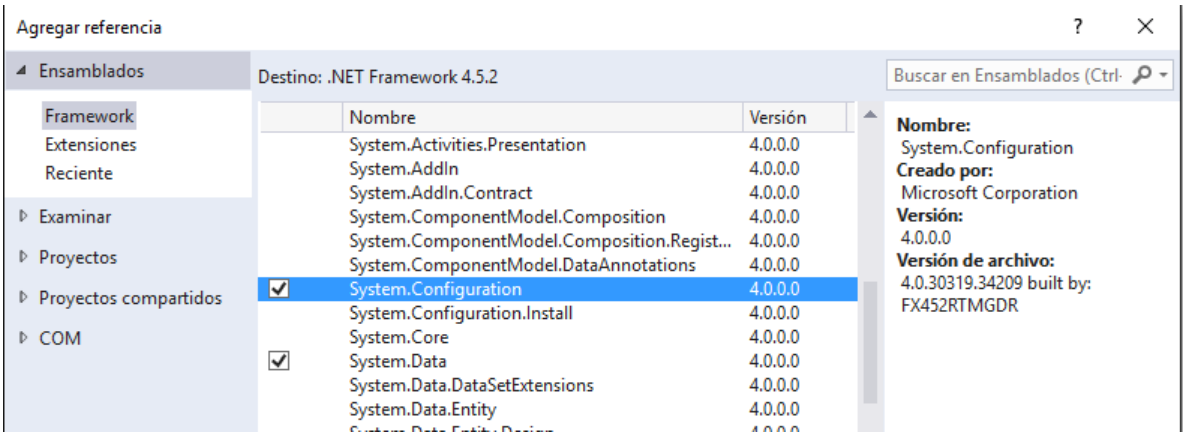

FIGURA 14

El fichero app.config contiene, nada más ser creado una estructura en formato xml para poder empezar a trabajar con él. Sobre esta estructura vamos a crear la sección "appSettings" y dentro de ella la pareja clave<->valor que va a representar la url de Arduino. Veamos el código:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
    <add key="ArduinoUrl" value="http://arduino.local/arduino/" />
   </appSettings>
</configuration>
```

A partir de aquí el objetivo es recuperar esta información en nuestra aplicación cuando sea necesaria y modificarla si la url de Arduino cambia de manera que quede almacenada para la siguiente ocasión. La forma de trabajar no es sencilla. En este enlace podemos encontrar no obstante una buena referencia, la cual hemos seguido para realizar los siguientes pasos. En primer lugar se deberán crear las estructuras necesarias para poder leer y escribir en el fichero de configuración en tiempo de ejecución.

Las clases y métodos que se van a emplear pertenecen al espacio de nombres:

```
using namespace System::Configuration;
```

Que habrá que habilitar en la ventana de configuración y también en la ventana principal si queremos acceder a estos ajustes desde ella. Asimismo vamos a declarar una serie de variables que pueden ser globales o locales del método que vaya a acceder a la configuración:

```
String^ exePath; //Ruta del fichero de configuración
System::Configuration::Configuration^ config; //Objeto que representa la
configuración de la aplicación
AppSettingsSection^ appSettingSection; //Sección de ajustes de la aplicación
```

El uso de estas variables que nos permite acceder a los ajustes de configuración es el siguiente:

```
exePath = Reflection::Assembly::GetExecutingAssembly()->Location;
config = ConfigurationManager::OpenExeConfiguration(exePath);
appSettingSection = (AppSettingsSection^)config->GetSection("appSettings");
```

En este momento el acceso está configurado. Conviene decir que suele ser recomendable añadir a continuación la siguiente sentencia:

```
ConfigurationManager::RefreshSection("appSettings");
```

Su misión es obligar a la lectura desde disco duro de los ajustes de configuración. De no hacerlo, los ajustes que se modifique en tiempo de ejecución no estarán disponibles para la aplicación hasta su siguiente ejecución, nunca en la actual.

Una vez realizada la configuración vamos a ver cómo leer y modificar los ajustes, concretamente en nuestro ejemplo cómo leer y escribir la url de Arduino. La forma de leer un ajuste es obtener su valor a partir de la clave:

```
String^ valor = ConfigurationManager::AppSettings["ArduinoUrl"];
```

El valor se devuelve como cadena de caracteres (String^). En caso de trabajar con otro tipo de valores, se puede hacer la conversión con posterioridad. El procedimiento para escribir es similar:

```
appSettingSection->Settings["ArduinoUrl"]->Value = "http://arduino.local/arduino/";
```

Para que el valor se traslade al fichero de configuración, en algún momento deberemos emplazar la sentencia:

```
config->Save();
```

Lo habitual es hacerlo al abandonar la ventana activa, ya sea en respuesta al evento FormClosing o en nuestro caso al pulsar el botón “Aceptar” de la ventana de configuración.

Después de realizados estos ajustes es interesante ver cómo ha quedado el código de respuesta al click en el botón de aceptar:

```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e)
{
  try {
       this->httpCliente->BaseAddress = gcnew Uri(textBox1->Text);
       Task<HttpResponseMessage^>^ responseTask = httpCliente-
>GetAsync("digital/13/0");
       HttpResponseMessage^ response = responseTask->Result;
       response->EnsureSuccessStatusCode();
       appSettingSection->Settings["ArduinoUrl"]->Value = textBox1->Text;
       config->Save();
       BaseAddress::Close();
       }
  catch (UriFormatException^) {
       textBox1->Text = "Formato de URl no válido";
       }
  catch (InvalidOperationException^) {
       textBox1->Text = "No se encuentra Arduino";
       }
  catch (AggregateException^) {
```

```
        textBox1->Text = "No se encuentra Arduino";
        }
}
```

Vemos que para comprobar que la dirección es correcta mandamos la orden de apagar el led conectado a la salida 13 que hemos encendido al inicio de la aplicación. De esta forma tenemos una confirmación también visual de que todo está correcto. Una vez comprobada la validez de la url introducida procedemos a guardarla en el fichero de configuración para que esté disponible en adelante. Previamente, en el constructor de la clase correspondiente a esta ventana de configuración habremos leído esta url y la habremos trasladado a la caja de texto:

```
BaseAddress(void)
{
        InitializeComponent();
        //
        httpCliente = gcnew HttpClient();
        exePath = Reflection::Assembly::GetExecutingAssembly()->Location;
        config = ConfigurationManager::OpenExeConfiguration(exePath);
        appSettingSection = (AppSettingsSection^)config->GetSection("appSettings");
        ConfigurationManager::RefreshSection("appSettings");
        textBox1->Text = ConfigurationManager::AppSettings["ArduinoUrl"];
        //
}
```

SALIDAS DIGITALES VÍA WEB

Una vez que hemos resuelto la forma de conexión, enviar órdenes a las salidas digitales es muy sencillo. De hecho ya hemos visto cómo hacerlo en una sola instrucción en el apartado anterior. Esta opción funciona correctamente, pero quizá no es la solución definitiva. Como hemos apuntado, la ejecución del método es asíncrona, lo que nos puede llevar a algún funcionamiento inesperado. Por otro lado, estos métodos asíncronos son proclives a generar excepciones, principalmente debido a la disponibilidad del destinatario. Esto debemos controlarlo adecuadamente.

Como salidas digitales nos la shield proporciona 4 diodos led (rojo, amarillo, verde, blanco) conectados a las salidas D11, D10, D9 y D6 respectivamente. Nuestra aplicación Arduino permite sin modificación entender y ejecutar las órdenes de encendido y apagado. Tan solo tendremos que asegurarnos de que hemos configurado las señales anteriores como salidas:

```
//Salidas digitales en la shield
pinMode(10, OUTPUT);
pinMode(11, OUTPUT);
pinMode(9, OUTPUT);
pinMode(6, OUTPUT);
```

Vamos a crear en nuestra interfaz botones biestables para encender y apagar los leds, todos excepto el blanco, que para hacerlo un poco diferente lo gobernaremos mediante un pulsador monoestable.

Los interruptores biestables se pueden incorporar desde el cuadro de herramientas de Visual Studio como “checkBox”. Ajustaremos sus propiedades “Text”, para hacerlo equivaler a la

señal que gobierna, “Appearence” que estableceremos como “Button” y “FlatStyle” que ajustaremos a “Popup”. El resultado se puede ver en la Figura 15.

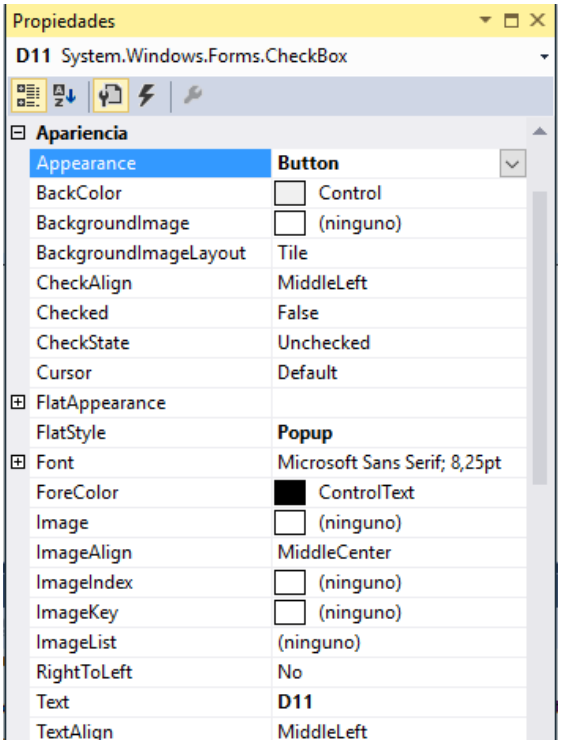

FIGURA 15

Procederemos de forma análoga para las 3 señales que queremos gobernar mediante interruptores biestables. Una vez incorporados los controles, haremos doble click sobre cada uno de ellos. Esto abrirá el método de respuesta al evento de cambio de su estado (en respuesta a un click de ratón). Este método será el encargado de enviar la orden. Su implementación no solamente va a incorporar la instrucción correspondiente, sino que se pretende que se enfrente también a las peculiaridades de este método. El código sería el siguiente:

```
private: System::Void D11_CheckedChanged(System::Object^  sender, System::EventArgs^
e) {
  if(D11->Checked){
    try {
      Task<HttpResponseMessage^>^ responseTask = httpArduino-
>GetAsync("digital/11/1");
       HttpResponseMessage^ response = responseTask->Result;
       response->EnsureSuccessStatusCode();
       D11->BackColor = Color::Red;
    }
    catch (InvalidOperationException^) {
       textBox1->Text = "No se encuentra Arduino";
    }
    catch (AggregateException^) {
       textBox1->Text = "No se encuentra Arduino";
    }
    }
    else {
```

```
        try {
          Task<HttpResponseMessage^>^ responseTask = httpArduino-
>GetAsync("digital/11/0");
          HttpResponseMessage^ response = responseTask->Result;
          response->EnsureSuccessStatusCode();
          D11->BackColor = BackColor;
        }
        catch (InvalidOperationException^) {
          textBox1->Text = "No se encuentra Arduino";
        }
        catch (AggregateException^) {
          textBox1->Text = "No se encuentra Arduino";
        }
    }
}
```

Vemos que el método de envío va seguido de dos instrucciones:

```
HttpResponseMessage^ response = responseTask->Result;
response->EnsureSuccessStatusCode();
```

Su misión es, en primer lugar extraer el objeto resultado de la consulta que se ha realizado, para luego asegurarse de que la operación se ha completado con éxito. De esta forma, lo que era una operación asíncrona se transforma en síncrona. Podemos comprobarlo si comparamos el resultado de la ejecución del código actual con el que obtendríamos si comentáramos la segunda de estas dos líneas. Veremos que en este caso, el color del botón, que lo cambiamos al color del led que gobierna cuando éste se ha activado, se actualiza inmediatamente, incluso antes de que se encienda el led afectado. Si recuperamos la línea de código eliminada, veremos que la secuencia de acontecimientos es justo al revés: primero se enciende el diodo y luego cambia el color del botón.

Finalmente señalar que conviene capturar las excepciones que se puedan producir si se pierde la comunicación con Arduino. En este caso hemos decidido mostrar la advertencia en un cuadro de texto: “textBox1”.

Con estos ajustes, el envío de órdenes a las salidas digitales de Arduino vía web resulta análogo a hacerlo por el puerto serie o por cualquier otro método.

ENTRADAS DIGITALES VÍA WEB

Trabajar con señales de entrada en esta arquitectura cliente-servidor introduce una diferencia fundamental respecto de hacerlo en forma de conexión punto a punto como es el puerto serie. La información en este caso se encuentra disponible en el servidor, pero es responsabilidad del cliente consultarla en el momento adecuado. El método que hemos empleado para enviar órdenes nos permite perfectamente realizar la consulta, ya que originalmente está pensado para ello, pero deberá ser nuestra aplicación la que tome la iniciativa. Vamos a colocar una serie de botones no accesibles que representen el estado de las entradas digitales de la shield y que corresponden con los pulsadores situados en las señales D12, D8, D7 y D4. Añadiremos también un botón que nos permita habilitar/deshabilitar la actualización. Enfocado de esta manera, el problema se vuelve análogo a la lectura del puerto serie. En este caso creábamos

una tarea (background worker) que permanecía en bucle infinito consultando el estado de la recepción del puerto. Vamos a seguir ahora el mismo procedimiento. Para ello vamos a incorporar los 5 botones mencionados y el componente invisible correspondiente al "background worker" (Figura 16).

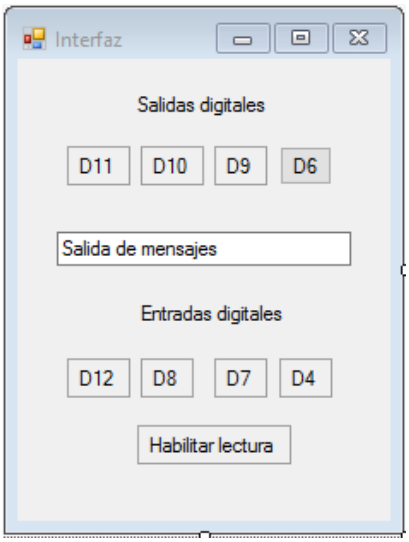

FIGURA 16

Hemos ubicado también una caja de texto para mostrar los mensajes que se reciben de Arduino, lo cual siempre es ilustrativo. El inicio y parada de la tarea en segundo plano exige que ésta permita la cancelación. Para ello deberemos ajustar su propiedad "WorkerSupportsCancellation" a "true". Una vez hecho así, codificaremos el método correspondiente al cambio de estado del botón de habilitación (que de nuevo es un control "checkBox" con apariencia de botón) de la siguiente manera:

```
private: System::Void HabilitarLectura_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	if (HabilitarLectura->Checked) {
		LeerEntradas->RunWorkerAsync();
		HabilitarLectura->BackColor = Color::Red;
	}
	else {
		LeerEntradas->CancelAsync();
		HabilitarLectura->BackColor = BackColor;
	}
}
```

SALIDAS ANALÓGICAS VÍA WEB

El envío de valores a las salidas analógicas no plantea realmente nuevos desafíos. Basta con sustituir la palabra "digital" por "analog" y lógicamente los valores "0" y "1" por valores enteros entre 0 y 255.

En primer lugar vamos a seleccionar un control en Windows Forms que nos permita establecer el valor de la salida. Lo más apropiado que encontramos es el denominado "TrackBar" que nos permite desplazar un cursor a lo largo de una barra adquiriendo valores comprendidos entre dos propiedades del objeto:

- "Minimum": que por defecto es cero y así lo dejaremos.
- "Maximum": que ajustaremos a 255.

El evento por defecto cuya gestión se abre si hacemos doble click en el control es el correspondiente a la acción "ValueChanged". Parece lógico aprovecharlo para informar a la placa Arduino de que debe cambiar el valor de la salida. Sin embargo esto no va a funcionar correctamente ya que el evento se lanza continuamente mientras estamos desplazando el cursos por la barra, no al dar por finalizado el recorrido. De esta forma se generan múltiples eventos de comunicación que llegan a saturar el buffer de recepción de Arduino. Utilizaremos en su lugar el evento "MouseUp", que se lanzará cuando liberemos el ratón dentro del control. Esto es lo que hacemos al terminar de arrastrar el cursos por la barra; lo ideal para poder informar solamente del valor final. Haremos doble click en este evento, como se ve en la Figura 17, para que sea incorporado el código correspondiente al método que lo gestiona.

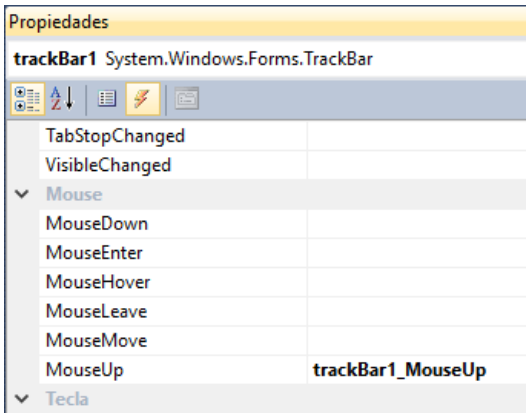

FIGURA 17

El método quedará programado de la siguiente manera:

```
private: System::Void trackBar1_MouseUp(System::Object^  sender,
System::Windows::Forms::MouseEventArgs^  e) {
       String^ a;
       a = AnalogOutput->Value.ToString(); //Transforma la posición del cursor en
string
       a = "analog/6/" + a; //Se añade al principio del string  (salida analógica)
       try {
              Task<HttpResponseMessage^>^ responseTask = httpArduino->GetAsync(a);
              HttpResponseMessage^ response = responseTask->Result;
              response->EnsureSuccessStatusCode();
       }
       catch (InvalidOperationException^) {
              textBox1->Text = "No se encuentra Arduino";
       }
       catch (AggregateException^) {
              textBox1->Text = "No se encuentra Arduino";
       }
```

```
}
```

ENTRADAS ANALÓGICAS VÍA WEB

Como hemos podido apreciar en el caso de las entradas digitales, la monitorización presenta cierto retardo y nos puede llevar a perder eventos que han sucedido en un tiempo menor que el que transcurre entre dos consultas. Trasladado esto al caso de las entradas analógicas, nos podemos encontrar con que la monitorización de un determinado parámetro físico resulta inviable. Para resolver esta limitación se impone una forma de trabajar diferente: dejemos que Arduino muestree las señales respetando el criterio de Nyquist-Shanon y leamos desde el PC la información en bloques. Para poder llevar esto adelante tendremos que enfrentarnos a dos aspectos nuevos:

- Almacenamiento de datos en fichero usando la tarjeta externa SD.
- Transferencia de ficheros inalámbrica.

Para usar la tarjeta SD como almacén de datos basta con conectarla. Como va a estar gobernada pode el procesador Atheros, para acceder a ella tendremos que emplear la notación de Linux "/mnt/sd/". Para probar esta funcionalidad podemos recurrir inicialmente a un ejemplo ya realizado. En este enlace encontramos una aplicación de "datalogger", esto es, de adquisición de datos, que guarda periódicamente el estado de varias entradas analógicas en un fichero emplazado en la tarjeta SD. ¿Cómo acceder a ese fichero? La forma más sencilla probablemente es descargarlo en el momento que uno quiera vía "scp". Para ello tendremos que instalar una aplicación. La más ampliamente utilizada es WinSCP. Es una aplicación muy sencilla de utilizar en la que podremos configurar y salvar la conexión con Arduino como se ve en la Figura 18

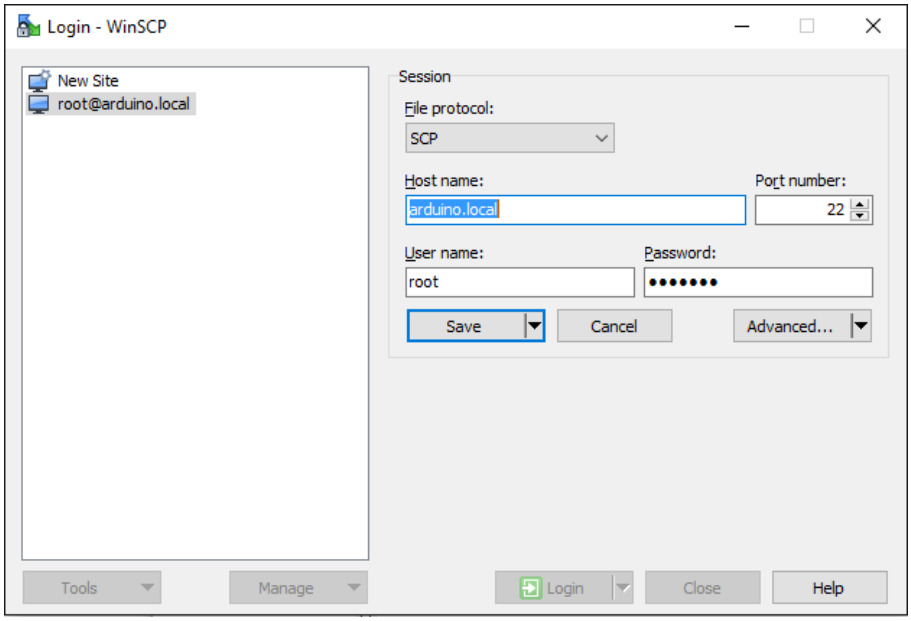

FIGURA 18

Una vez establecida la conexión veremos un doble explorador de archivos: a la izquierda el árbol de directorios de la máquina local; a la derecha el de la tarjeta SD como se en la Figura 19.

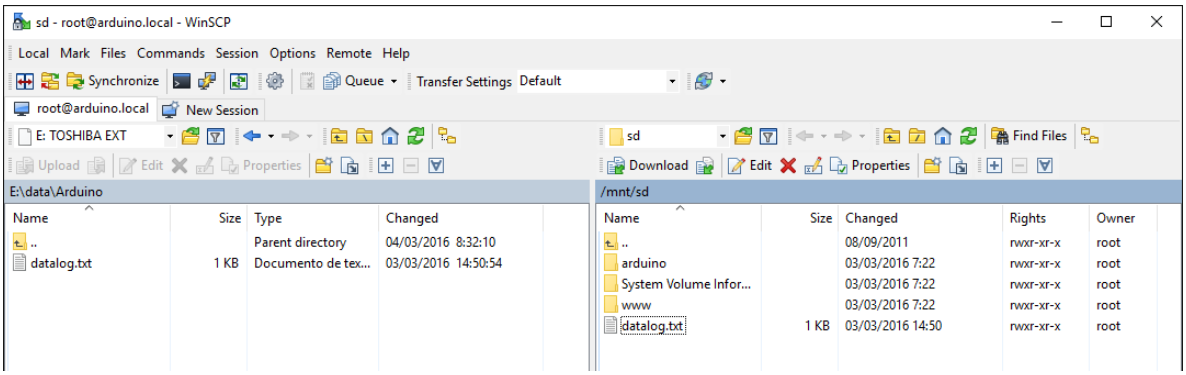

FIGURA 19

Mediante el botón “Download” podemos descargar el archivo de datos a nuestro ordenador. De esta forma podemos recuperar en nuestra máquina local la información almacenada por Arduino sin extraer la tarjeta SD e incluso sin impedir que Arduino siga almacenando en ella.

En este enlace podemos encontrar información más detallada acerca del funcionamiento de WinSCP.

La monitorización de estas entradas es similar a la adquisición de señales digitales. En este caso vamos a emplear controles del tipo “trackBar” para representar el estado de los potenciómetros que en la shield están conectados a las entradas analógicas 0 y 1. Ajustaremos su propiedad “Maximum” a 1023 (ya que los conversores analógico-digital son de 10 bits), y la propiedad “Enabled” a False (para que no puedan ser manipulados desde la interfaz.

Habilitaremos un hilo específico (BackgroundWorker) para la lectura asíncrona de las señales. También situaremos un “checkBox” con apariencia de botón como en el caso de las entradas digitales para habilitar/deshabilitar la lectura. La actualización del estado de los potenciómetros tendrá que realizarse mediante invocación ya que se encuentran declarados en el hilo principal. Para ello en primer lugar debemos declarar la delegación en la zona privada de la clase principal:

```
delegate void SetPot1Delegate(int valor); //Delegado para acceso seguro a Pot1
delegate void SetPot2Delegate(int valor); //Delegado para acceso seguro a Pot1
```

El código correspondiente al cambio de estado del control de habilitación es análogo al empleado en las señales digitales:

```
private: System::Void LecturaAnalogica_CheckedChanged(System::Object^  sender,
System::EventArgs^  e) {
	if (LecturaAnalogica->Checked) {
		LeerAnalogicas->RunWorkerAsync();
		LecturaAnalogica->BackColor = Color::Red;
	}
	else {
		LeerAnalogicas->CancelAsync();
		LecturaAnalogica->BackColor = BackColor;
```

```
	}
}
```

El contenido del hilo de lectura de las señales analógicas será el siguiente:

```
private: System::Void LeerAnalogicas_DoWork(System::Object^  sender,
System::ComponentModel::DoWorkEventArgs^  e) {
	String^ cadena;
	int valor;
	while (!this->LeerAnalogicas->CancellationPending) {
		try {
			Task<HttpResponseMessage^>^ responseTask = httpArduino-
>GetAsync("analog/0");
			HttpResponseMessage^ response = responseTask->Result;
			response->EnsureSuccessStatusCode();
			String^ respuesta = response->Content->ReadAsStringAsync()->Result;
			SetText(respuesta);
			cadena = respuesta;
			cadena = cadena->Substring(20); //Donde empieza el dato
			SetPot1(Int32::Parse(cadena));
		}
		catch (InvalidOperationException^) {
			textBox1->Text = "No se encuentra Arduino";
		}
		catch (AggregateException^) {
			textBox1->Text = "No se encuentra Arduino";
		}
		try {
			Task<HttpResponseMessage^>^ responseTask = httpArduino-
>GetAsync("analog/1");
			HttpResponseMessage^ response = responseTask->Result;
			response->EnsureSuccessStatusCode();
			String^ respuesta = response->Content->ReadAsStringAsync()->Result;
			SetText(respuesta);
			cadena = respuesta;
			cadena = cadena->Substring(20);
			SetPot2(Int32::Parse(cadena));
		}
		catch (InvalidOperationException^) {
			textBox1->Text = "No se encuentra Arduino";
		}
		catch (AggregateException^) {
			textBox1->Text = "No se encuentra Arduino";
		}
	}
}
```

La operación implica la llamada al método de actualización, mediante invocación, del estado del control. Veamos el correspondiente al potenciómetro 1, ya que el 2 es totalmente análogo:

```
private: void SetPot1(int valor) {
	if (this->Pot1->InvokeRequired) {
		SetPot1Delegate^ d = gcnew SetPot1Delegate(this, &MyForm::SetPot1);
		this->Invoke(d, gcnew array<Object^> { valor });
	}
```

```
        else {
                this->Pot1->Value = valor;
        }
}
```

Dato que tenemos nuestro Arduino Yun conectado vía wifi, nos podemos plantear acceder a él y a su información mediante otros dispositivos. En este apartado vamos a ver cómo podemos crear una aplicación Android que nos permita hacer básicamente los que hemos hecho desde el PC en el apartado anterior, desde nuestro teléfono móvil o Tablet Android.

Entornos de desarrollo Android existen en gran número. En este momento nos vamos a centrar en un entorno que podríamos denominar "amateur" con el fin de no extender demasiado el tamaño y complejidad de este apartado. Se trata de App Inventor, un entorno web desarrollado por el MIT (Massachusets Insitute of Technology) en colaboración con Google. Para poder acceder solamente es necesario disponer de una cuenta de Google.

Accederemos pro tanto a la versión más reciente: App Inventor 2 y comenzaremos el desarrollo de nuestra aplicación. El objetivo, obtener una interfaz con Arduino Yun análoga a la que hemos desarrollado mediante VicualC++ para el PC.

La interfaz de programación se compone de dos vistas del programa: la de interfaz ("Designer") y la de código ("Blocks"). Podemos movernos de una a otra mediante los botones que se muestran en la Figura 20.

FIGURA 20

En este tipo de entorno, en el que las herramientas de edición son tan limitadas es conveniente acertar de entrada con el diseño de la aplicación. Dado que tratamos de reproducir la funcionalidad de la aplicación del PC, puede ser una buena idea contar con 4 pantallas para las distintas funcionalidades:

- Entradas digitales
- Salidas digitales
- Entradas analógicas
- Salidas analógicas

Para poder seleccionar una de ellas será necesario tener una pantalla de bienvenida. Nos vamos entonces a un diseño de 5 pantallas. La gestión de pantallas se realiza fácilmente mediante los botones que se muestran en la Figura 21. Conviene destacar que la pantalla de bienvenida mantendrá siempre el nombre por defecto (Screen1), mientras que el resto pueden recibir el nombre que deseemos.

FIGURA 21

Comenzaremos por el diseño de la pantalla de bienvenida. Va a constar de:

- Dos etiquetas

- Una caja de texto
- 5 botones

Todo ello lo podemos encontrar dentro de la vista de Diseño (“Designer”) en el menú “User Interface” como se muestra en la Figura 22.

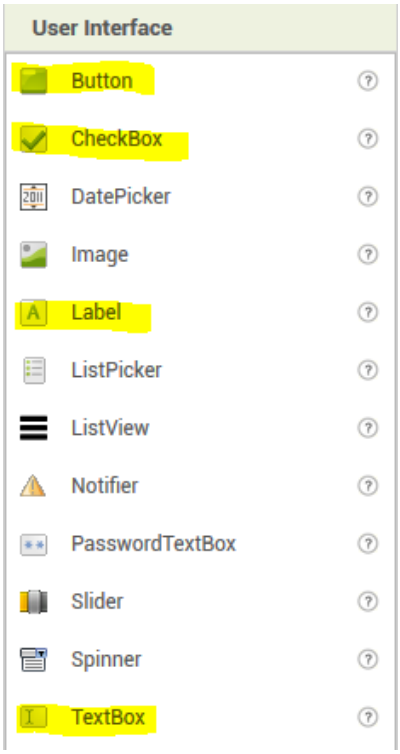

FIGURA 22

La caja de texto la utilizaremos para introducir la dirección IP de nuestro Arduino Yun y también para visualizar la respuesta al intento de conexión. Encima y debajo de ella pondremos las etiquetas que señalen estas circunstancias.

4 de los 5 botones nos permitirán seleccionar el tipo de señales con las que queremos operar y, por tanto, a qué otra pantalla queremos acudir. Para ubicarlos correctamente podemos utilizar un modelo de ordenación de los disponibles en el menú “Layout” como se puede ver en la Figura 23.

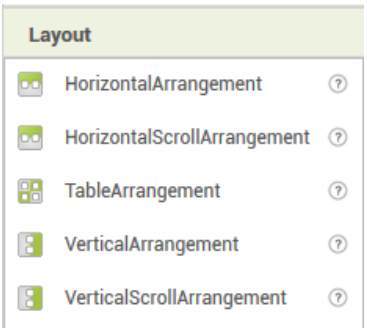

FIGURA 23

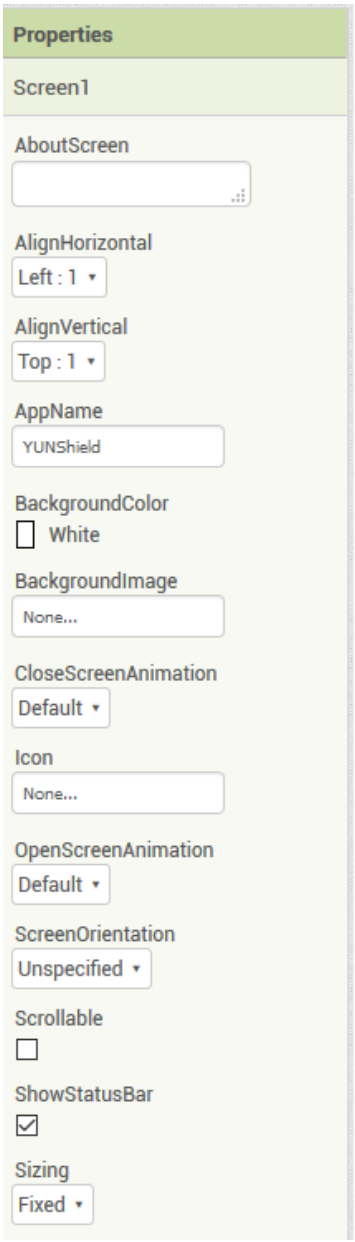

FIGURA 24

Mediante el menú "Properties" de la derecha (Figura 24) es posible ajustar de una forma sencilla todo lo que tiene que ver con cada elemento: color, tamaño, forma, texto, etc.

En la propia Figura 24 se pueden ver algunas de las propiedades de la pantalla, que también es un elemento configurable.

Vemos cómo se pueden determinar aspectos tales como si la pantalla va a ser vertical u horizontal, el icono asociado o si admite desplazamiento ("Scroll").

Para concluir el diseño de esta pantalla de bienvenida, introduciremos dos elementos que no son visibles para el usuario, pero sí son necesarios para la programación:

- Dentro del menú "Storage", seleccionaremos "TinyDB". Se trata de una pequeña base de datos local en la que se pueden almacenar valores de manera persistente, bien sea para ser compartidos por diferentes pantallas o bien para que estén disponibles en el momento del arranque de la aplicación.
- Dentro del menú "Conectivity" seleccionaremos "Web". Se trata de una herramienta que nos va a permitir intercambiar información en la red mediante el uso de servicios web.

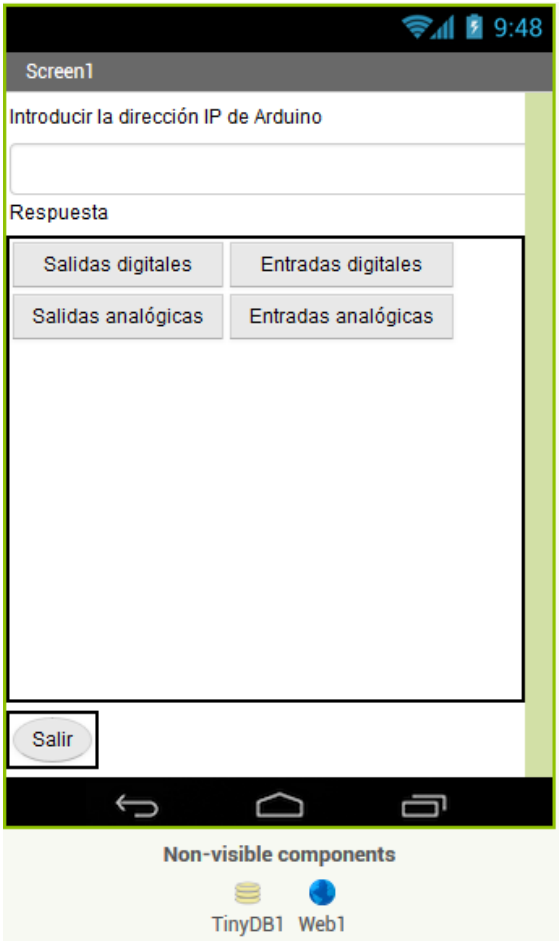

FIGURA 25

La Figura 55 es un ejemplo de cómo pueden quedar ubicados los distintos elementos en pantalla.

Podemos reubicarlos y modificar su aspecto como queramos. En todo caso, con estos elementos en pantalla estaremos preparados para pasar a desarrollar el código.

Para ello pulsaremos el botón "Blocks" para entrar en el lienzo de desarrollo.

En el menú de bloques ("Blocks") de la derecha podemos ver las herramientas disponibles. Existen herramientas de propósito general ("Built-in") y herramientas asociadas a cada uno de los elementos que hemos introducido en la vista de diseño de interfaz. Pulsando sobre cada uno de ellos se despliega el menú de opciones disponibles.

La primera funcionalidad que vamos a incluir es la de inicialización. Si pulsamos en "Screen1", se nos desplegarán sus métodos, entre los cuales tenemos "when Screen1 .Initialize". Este método nos permite establecer qué se ha de hacer en el momento de inicializar la pantalla. La Figura 26 muestra lo que hemos propuesto para esta aplicación.

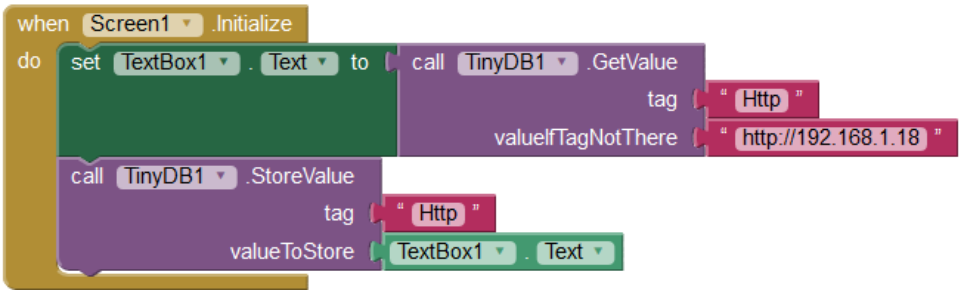

FIGURA 26

Traducido a texto lo que se hace es inicializar la caja de texto a la dirección IP en la que encontramos Arduino. Se espera que esta dirección esté almacenada en la base de datos local bajo la etiqueta "Http". La primera vez no estará inicializada, por lo que introducimos un valor

por defecto, en este caso http://192.168.1.18. A continuación este valor u otro que hubiera introducido el usuario queda almacenado para las siguientes inicializaciones.

Los botones de acceso a las 4 pantallas de gestión de señales se tratan de forma análoga. En la Figura 27 vemos el ejemplo de acceso a la pantalla de salidas digitales.

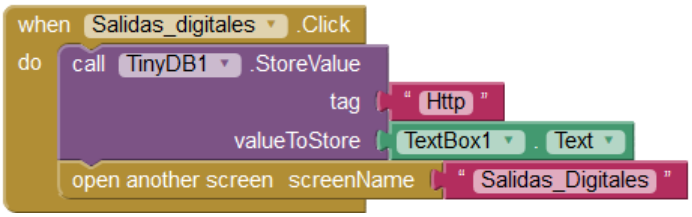

FIGURA 27

Cuando se pulsa el botón se guarda lo que el usuario hubiera introducido como dirección de Arduino y se abre la ventana correspondiente. Para terminar nos queda por codificar el botón de salida, que será la salida de la aplicación. Lo podemos ver en la Figura 28.

FIGURA 28

ENTRADAS DIGITALES

La adquisición de información de señales desde Android es conceptualmente igual que la desarrollada para el PC. Vamos a ver, no obstante, la forma de codificarlo en el entorno que estamos utilizando. Obviamente se pueden seguir diferentes estrategias. En este ejemplo lo que vamos a hacer es ir interrogando de forma cíclica a Arduino acerca del estado de sus entradas digitales.

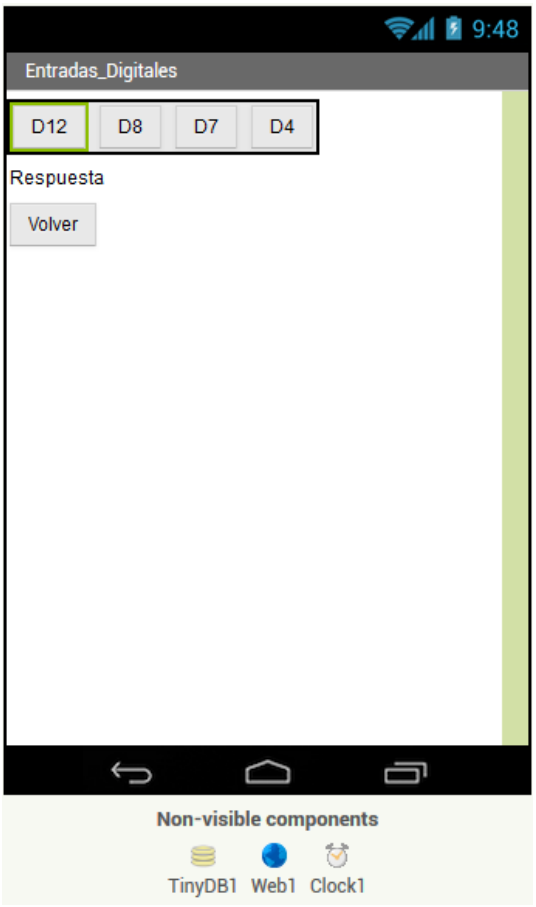

FIGURA 29

Para el diseño de la interfaz vamos a necesitar 4 botones, que situaremos en un arreglo horizontal y que corresponden con las 4 entradas digitales: D4, D7, D8, D12. Una etiqueta que situaremos debajo para indicar que los botones representan la respuesta de Arduino y un quinto botón para volver a la pantalla principal. Todo ello lo podemos ver en la Figura 59.

Vemos que adicionalmente tenemos que añadir la base de datos que ya teníamos creada, la herramienta web para poder acceder a Arduino y un reloj temporizador que será el que secuencia el acceso a las diferentes señales de Arduino, ya que se ha de hacer una a una.

En la Figura 30 vemos la inicialización de la variable que va a gobernar la secuencia de lectura junto con la habilitación del temporizador al iniciar la pantalla y la programación del botón de retorno.

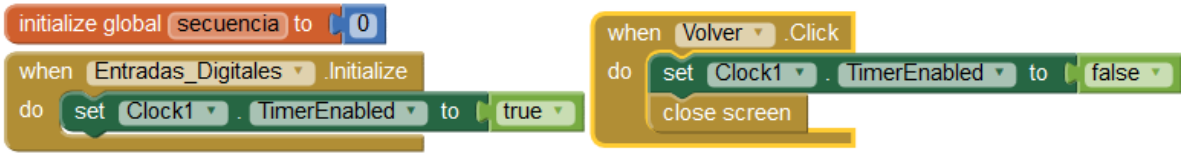

FIGURA 30

La operación de lectura se realiza mediante direcciones web que corresponden con dada una de las señales. Arduino responderá con el estado de cada una de ellas. Esto da lugar a dos partes diferenciadas en nuestra codificación:

- Programación de la secuencia temporizada de lectura de las entradas.
- Recogida e interpretación de la respuesta que proporciona Arduino.

La Figura 31 muestra la codificación de la secuencia. Cada vez que se produce la señal del temporizador, la secuencia se incrementa y se envía la dirección web de la entrada digital correspondiente. Para ello lo que hacemos es concatenar (método "join") la dirección base de Arduino con la secuencia correspondiente a cada señal.

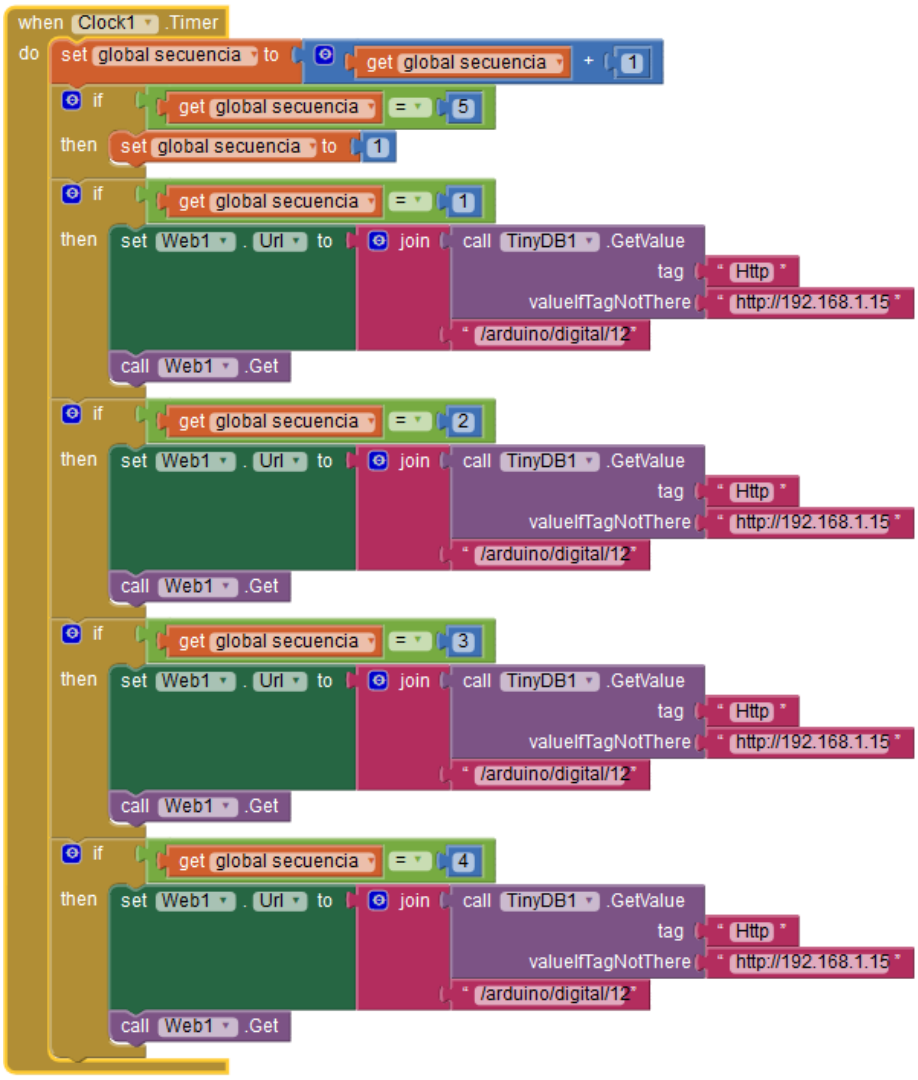

FIGURA 31

Si todo es correcto en el lado de Arduino, cada consulta irá seguida de una respuesta cuyo texto habremos configurado en el sketch. En nuestro caso es “Pin xx set to x”. Tendremos que ve en cada caso qué respuesta se ha recibido y programar la acción correspondiente. Una opción la vemos en la Figura 32. Se trata de una comparación de texto literal y, según la respuesta cambiamos el color del botón correspondiente a la entrada aludida.

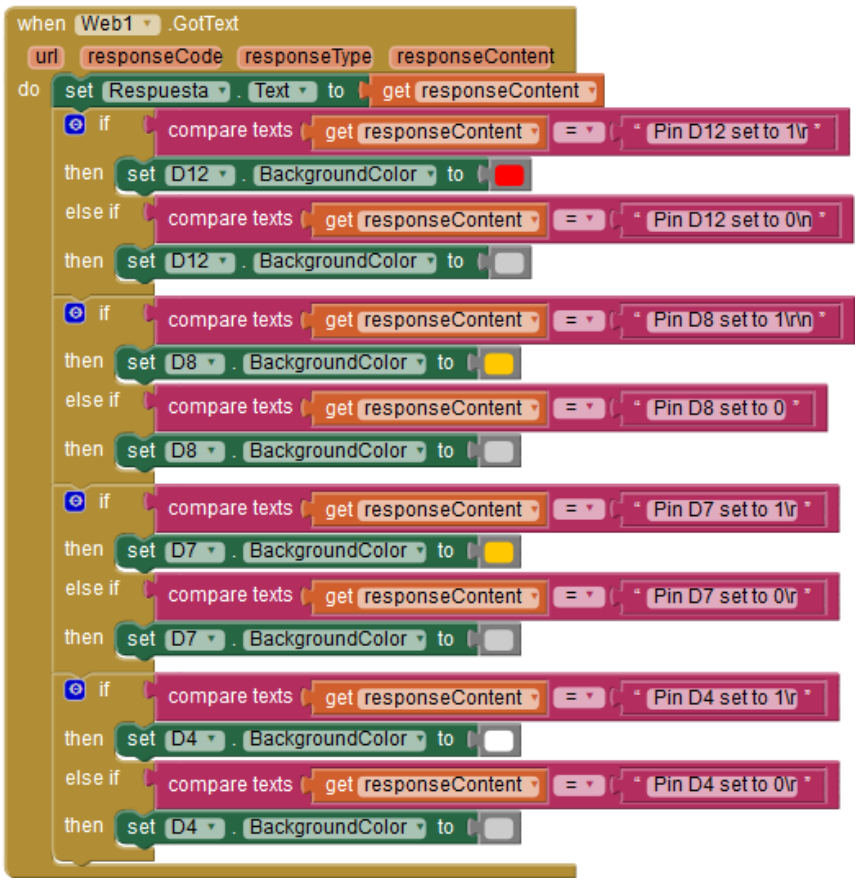

FIGURA 32

SALIDAS DIGITALES

Se trata de la opción sin duda más sencilla. Las órdenes se van a enviar a voluntad del usuario que pulsará el botón correspondiente a la salida que desea activar. Dado que Arduino va a proporcionar una respuesta en la que reflejará el estado de la señal modificada, se puede tomar esta respuesta como confirmación y modificar el color de fondo del botón correspondiente tal y como se hizo en la pantalla de entradas digitales.

Las salidas digitales se pueden tratar como monoestables o como biestables. Vamos a ver ambos casos, para lo cual consideraremos la salida D11 como monoestable y la D6 como biestable.

Que una salida sea monoestable implica que cambiará de estado al pulsar el botón y al liberarlo retornará a su estado original. En la Figura 33 se muestra cómo codificarlo.

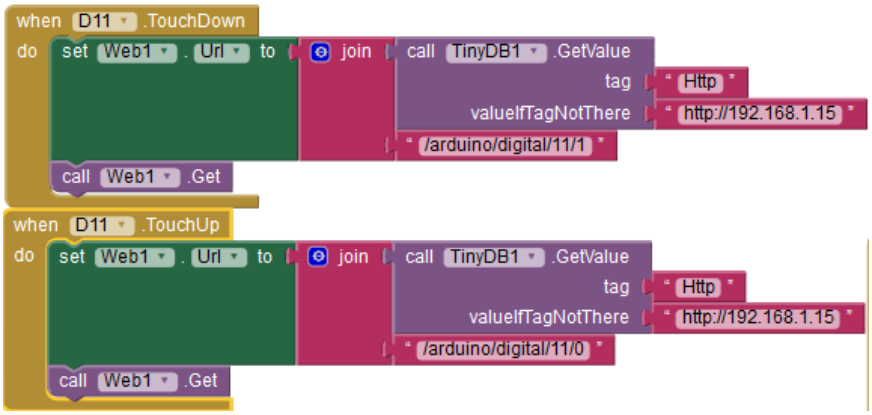

FIGURA 33

La agilidad de funcionamiento va a depender de los tiempos de reacción de cada uno de los elementos implicados, por lo que puede ser necesario ejecutar las órdenes con cierta lentitud para poder observar la respuesta esperada.

Una señal biestable cambiará de estado cuando se le dé la orden y lo mantendrá. En la Figura 34 vemos cómo conseguir este funcionamiento. En este caso, cada vez que se pulse D6 cambiará al estado contrario al que tuviera. De partida se inicializa el estado a "false".

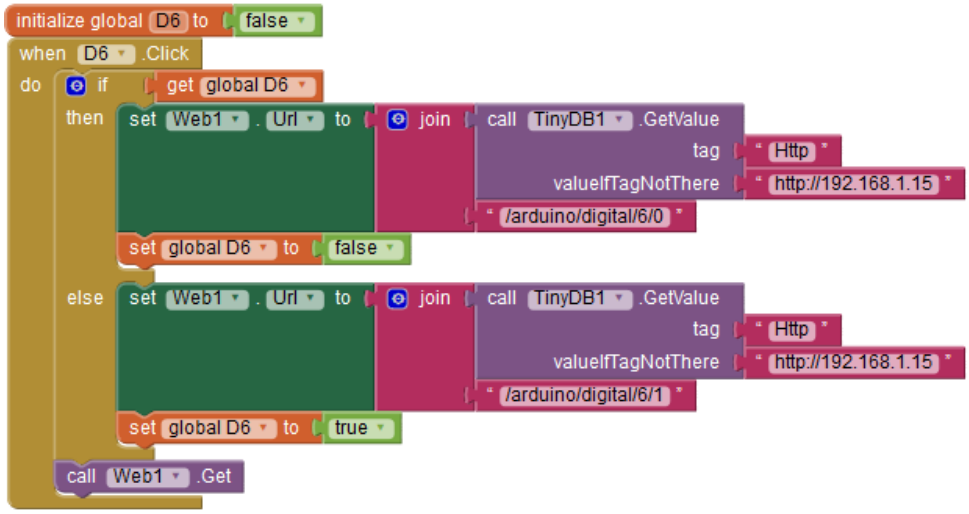

FIGURA 34

SALIDAS ANALÓGICAS

La gestión de las salidas analógicas es muy similar a la de las salidas digitales. En lugar de utilizar un botón para gobernarlas emplearemos una barra de deslizamiento "Slider" disponible en el menú "User Interface" de la vista de diseño. La posición de la barra se enviará a Arduino como valor de la señal analógica correspondiente. En la Figura 35 podemos ver la forma de hacerlo.

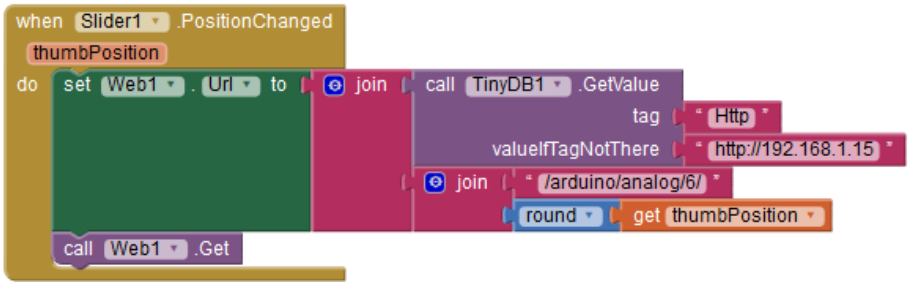

FIGURA 35

ENTRADAS ANALÓGICAS

La forma de gestionar estas señales es similar a la que nos permite gestionar las entradas digitales. Esto es así debido al modelo consulta/respuesta que nos proporciona Arduino. Podemos por tanto aprovechar el modelo de consulta secuencial que ya hemos empleado. Por ello nos centraremos en cómo tratar los datos analógicos que, ciertamente, presentan algunas peculiaridades sobre los digitales.

En nuestro ejemplo hemos incluido 3 señales analógicas: los dos potenciómetros, que representamos mediante barras de desplazamiento "Slider" y el sensor de temperatura, cuya medida vamos a representar gráficamente a lo largo del tiempo.

En la Figura 36 vemos cómo se tratan las respuestas de Arduino. Tenemos distintas llamadas a métodos que tratan cada una de las señales. En la parte de la derecha se observa cómo se ajusta al valor recibido a los límites de la barra de desplazamiento que representa el estado de uno de ls potenciómetros.

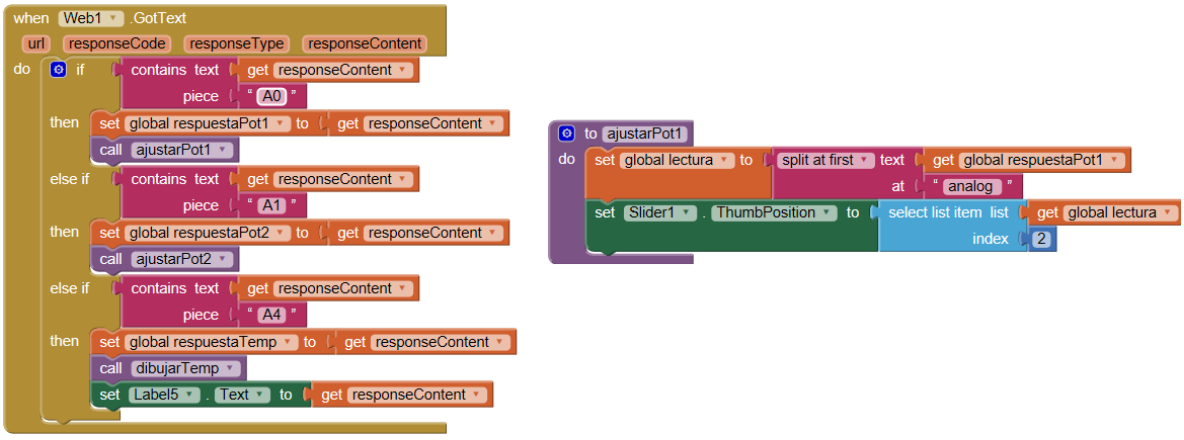

FIGURA 36

El método "ajustarPot1" desarma la cadena de caracteres devuelta por Arduino para extraer el valor numérico y transformarlo en la posición de la barra de desplazamiento. Como sabemos que Arduino nos va a proporcionar un valor entre 0 y 1023, solamente tenemos que ajustar los límites de la barra de desplazamiento a esos valores en la vista de diseño.

El tratamiento de la temperatura es más complicado, ya que nos hemos empeñado en representar gráficamente su evolución y AppInventor no nos proporciona herramientas de representación gráfica. Hemos construido un método para ello. En la Figura 37 vemos el cuerpo principal de "dibujarTemp".

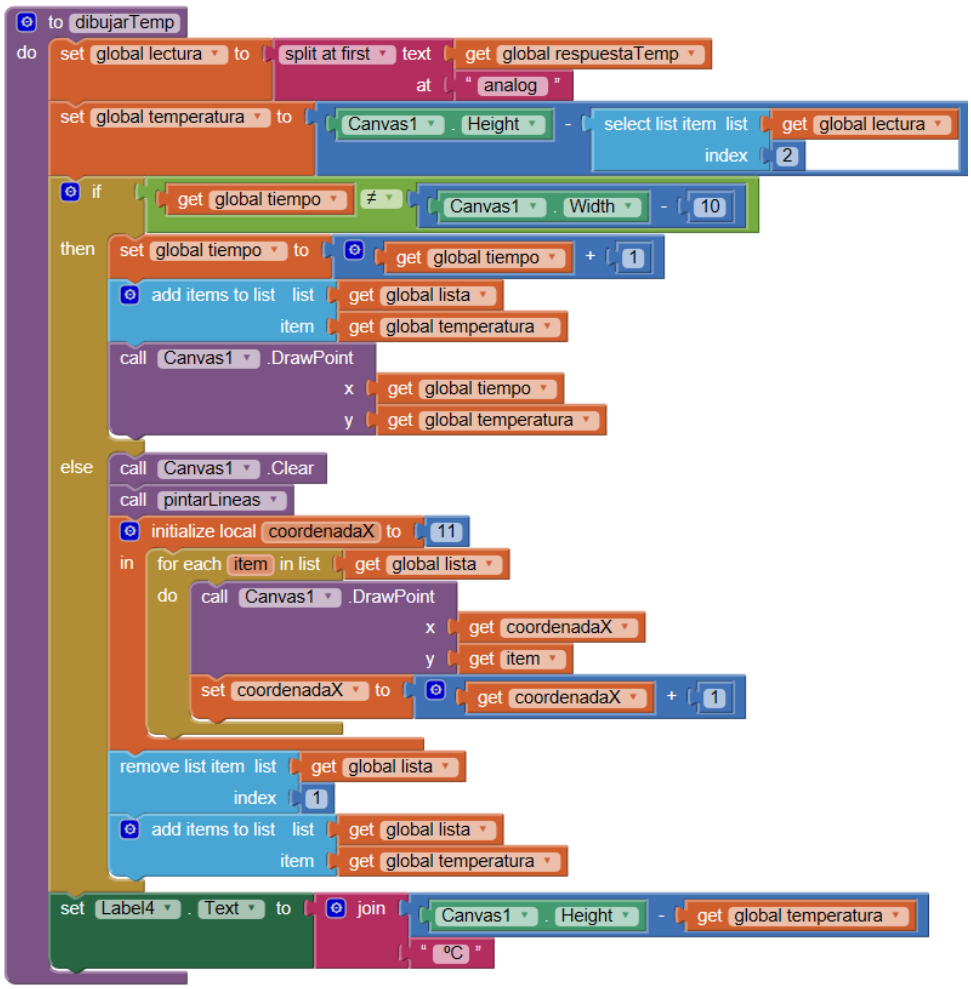

FIGURA 37

En la parte inicial tenemos la extracción del valor analógico al igual que en el caso del potenciómetro. Este valor se va a transformar en la coordenada vertical de la representación gráfica aunque, como la altura se mide desde la parte superior, se restará del tamaño total del lienzo de dibujo. La coordenada horizontal se irá incrementando cada vez que se dibuje un punto, de manera que la gráfica irá avanzando de izquierda a derecha. El problema viene cuando se alcanza el final del lienzo por el lado derecho. Cuando la representación llega al tamaño horizontal del lienzo menos 10 puntos, la forma de representación cambia. A partir de aquí, la gráfica se redibuja completa cada vez, incluidas las líneas de referencia. Cada vez que llega un nuevo dato, se elimina el más antiguo. Esto es un proceso tedioso, pero en un dispositivo con prestaciones medias no es percibido por el usuario que ve cómo la gráfica avanza en tiempo real.

En la Figura 38 vemos el método creado para dibujar los ejes de coordenadas. Esta operación se realiza en la inicialización de la pantalla y, como acabamos de comentar, cada vez que llega un nuevo dato una vez completada la gráfica a lo ancho.

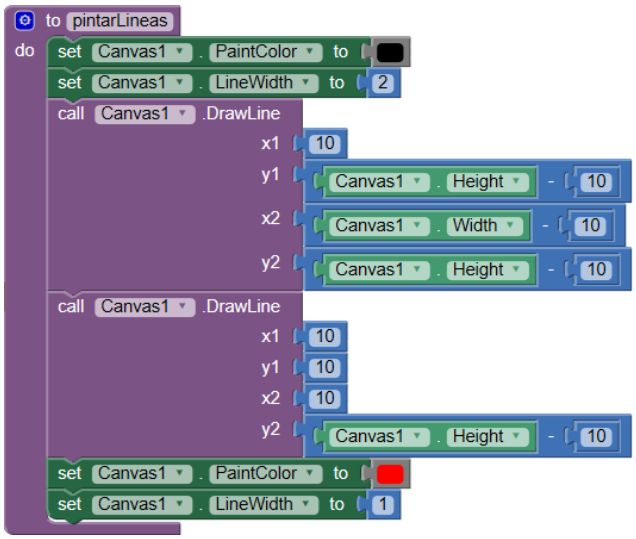

FIGURA 38

App Inventor nos proporciona los menús “Connect” y “Build” para depurar y construir la aplicación respectivamente. Podemos depurarla en nuestro dispositivo Android (“AI Companion”) conectado de forma remota o en un emulador instalado localmente. Una vez terminado el desarrollo, se puede generar el fichero instalable (.apk) y volcarlo a nuestro dispositivo, bien como fichero, bien mediante un QR.

La funcionalidad que nos proporciona Arduino Yun gracias a su conectividad nos abre la puerta a multitud de posibilidades como ya hemos visto. Sin embargo, una vez que tenemos nuestro sistema funcionando y con una interfaz inalámbrica nos podemos plantear que no es suficiente. Nos gustaría poder interactuar con nuestros dispositivos de manera remota.

Para que esto sea posible, nuestro Arduino Yun tendría que ser visible desde cualquier punto a través de Internet. Deberíamos poder acceder a la información generada por los sensores y también ser capaces de enviar órdenes a los actuadores.

Sin embargo, bien sea mediante la red wifi creada por Yun, o conectándolo a nuestro router, un dispositivo externo no va a poder localizar el dispositivo en la red. Esto se debe a que la dirección IP que empleamos no es pública. Se trata de una dirección privada que nos asigna temporalmente nuestro proveedor de servicios de internet (ISP) y que varía constantemente. De esta forma, tanto la aplicación de PC como la de Android, si no está ejecutándose en dispositivos conectados a la misma red que Arduino, no van a poder localizarlo y por lo tanto no nos van a dar ninguna funcionalidad.

Para solventar esta limitación vamos a necesitar ayuda, más concretamente vamos a necesitar disponer de un intermediario que haga que nuestro dispositivo se haga visible al resto del mundo. Esto lo podemos plantear de dos maneras:

- Utilizar un servicio de "relay" que haga directamente lo que hemos dicho: hacer visible nuestro dispositivo para el resto del mundo. En este sentido podemos acudir a proveedores como Yaler. Es quizá la opción más sencilla, pero no hemos localizado ninguno de estos proveedores que nos proporcione un acceso gratuito por tiempo ilimitado por lo que la hemos descartado para este trabajo.
- Utilizar un servidor en la nube a través del cual intercambiar información y comandos. Es la opción elegida ya que sí existen proveedores en el mercado que nos proporcionan un servicio gratuito, limitado pero indefinido. Desde luego suficiente para este trabajo.

En nuestro caso trabajaremos con Hostinger, aunque existen otros proveedores similares. Nos podemos dar de alta para el servicio gratuito creando una cuenta o simplemente utilizando nuestra cuenta de Google.

A continuación vamos a explicar cómo crear la infraestructura necesaria en Hostinger para poder trabajar con y contra Arduino. Una buena referencia para todo lo que vamos a explicar se puede encontrar en esta serie de vídeos que lo presentan de una forma muy dinámica.

Fundamentalmente vamos a necesitar dos elementos en nuestro alojamiento en la nube:

- Una base de datos en la que almacenar la información, tanto entrante como saliente.
- Una página web a través de la cual acceder a esa información desde Internet.

Ambas funcionalidades nos las proporciona Hostinger, así que en los siguientes apartados vamos a ver cómo ponerlas en funcionamiento. En la Figura 39 podemos ver un esquema de la arquitectura que estamos generando:

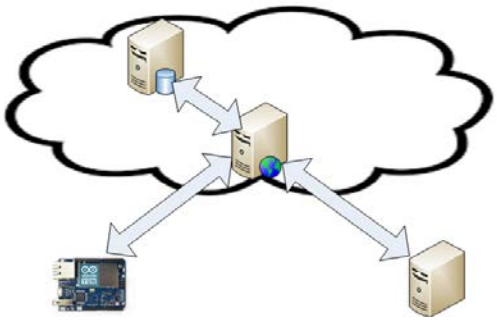

FIGURA 39

El intercambio de información entre Arduino y el usuario final se va a realizar a través de la intermediación del servidor web que a su vez interactuará con la base de datos en la que se encuentra almacenada la información.

En las opciones de conectividad precedentes hemos asumido que arduino y el PC o el dispositivo Android se encontraban en la misma red. En este caso no es así. Arduino se conectará a una red que le proporcione acceso a internet; así el usuario podrá interactuar con él desde cualquier navegador independientemente de su ubicación física. La conexión de arduino a Internet a través de una red wifi doméstica es trivial. En cuanto entremos dentro del alcance de la red deseada, como se puede observar en la Figura 4, tendremos la posibilidad de seleccionar la red e introducir la contraseña. A partir de ese momento, arduino se conectará a ella y para poder acceder a él tendremos que hacerlo desde dispositivos vinculados a la misma red. Con esa condición, toda la funcionalidad descrita hasta ahora en este documento seguiría siendo operativa. A partir de ahora, a través de nuestro router doméstico arduino será "accesible" además desde fuera de nuestro ámbito, eso sí, empleando el alojamiento externo como intermediario.

Una de las herramientas que nos proporciona Hostinger es la de creación y gestión de una base de datos MySQL. Podemos acceder a ella a través del icono que nos aparece en la página principal (Tablero) de nuestra cuenta (Figura 40).

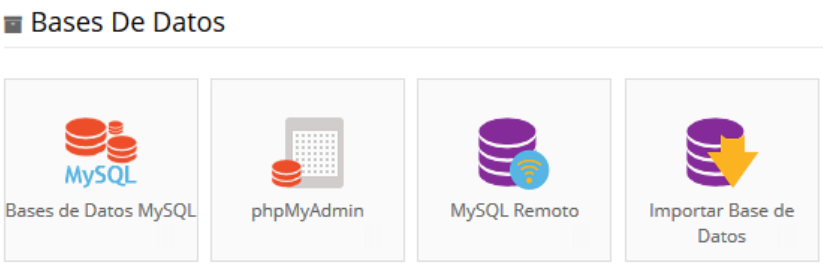

FIGURA 40

Al pulsar en él, entraremos en la página de creación y gestión de bases de datos MySQL. En la Figura 41 se muestran los campos que hay que rellenar para crear una nueva base de datos:

- Nombre de la base de datos: le llamaremos "Yun".
- Usuario: "Ardu".
- Contraseña: pondremos la que mejor nos parezca, por ejemplo "ArduinoYun".

FIGURA 41

Cuando pulsemos "Crear" veremos que en la lista de Bases de Datos y Usuarios MySQL Actuales que aparece en la parte inferior de la Figura 41 se ha añadido la nueva base de datos creada. Para administrarla recurriremos a la herramienta "phpMyAdmin" disponible en el

menú de la derecha de la misma página creación y también en el "Tablero" como se mostraba en la Figura 40. Nos da entrada a la página que vemos en la Figura 42.

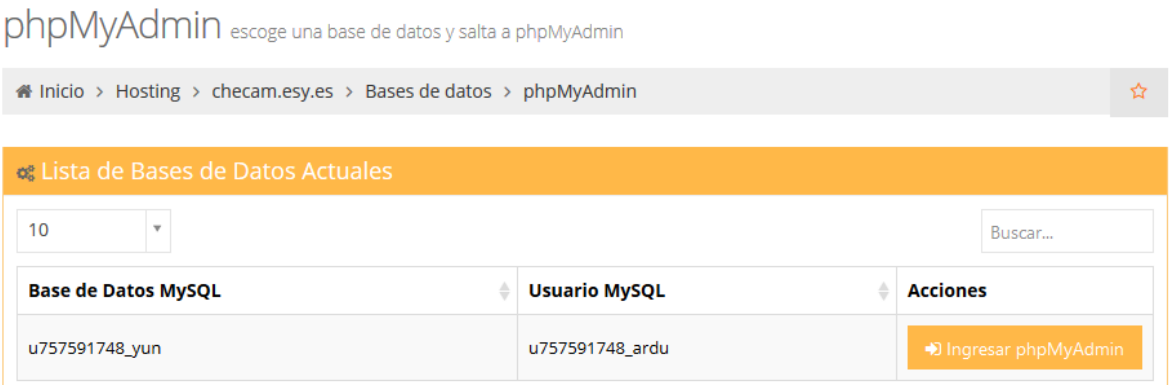

FIGURA 42

Cuando pulsemos en "Ingresar phpMyAdmin" accederemos a la funcionalidad de crear tablas columnas dentro de tablas. Como habitualmente estamos dividiendo las señales en 4 categorías, crearemos otras tantas tablas:

Tipo de señal	Nobre de Tabla	Columnas
Entradas analógicas	EntAn	4 (Pot1, Pot2, Infrarrojos, Temperatura)
Entradas digitales	EntDi	4 (D4, D7, D8, D12)
Salidas analógicas	SalAn	1 (LED blanco)
Salidas digitales	SalDi	4 (D6, D9, D10, D11)

En la Figura 43 vemos la creación de la primera tabla.

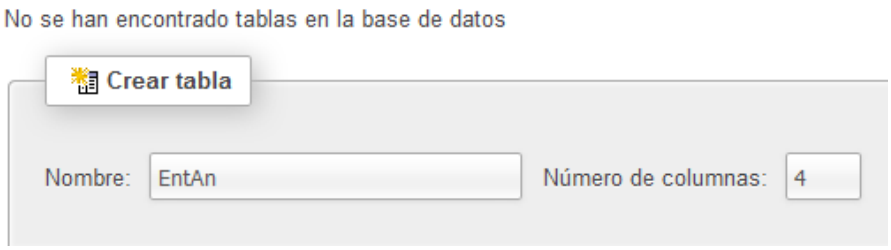

FIGURA 43

Cuando pulsemos "Continuar" entraremos en la ventana de configuración de la tabla que aparece en la Figura 44. En ella hemos configurado las cuatro entradas, dos de ellas (los potenciómetros) como datos enteros y las otras dos (infrarrojos y temperatura) como flotantes.

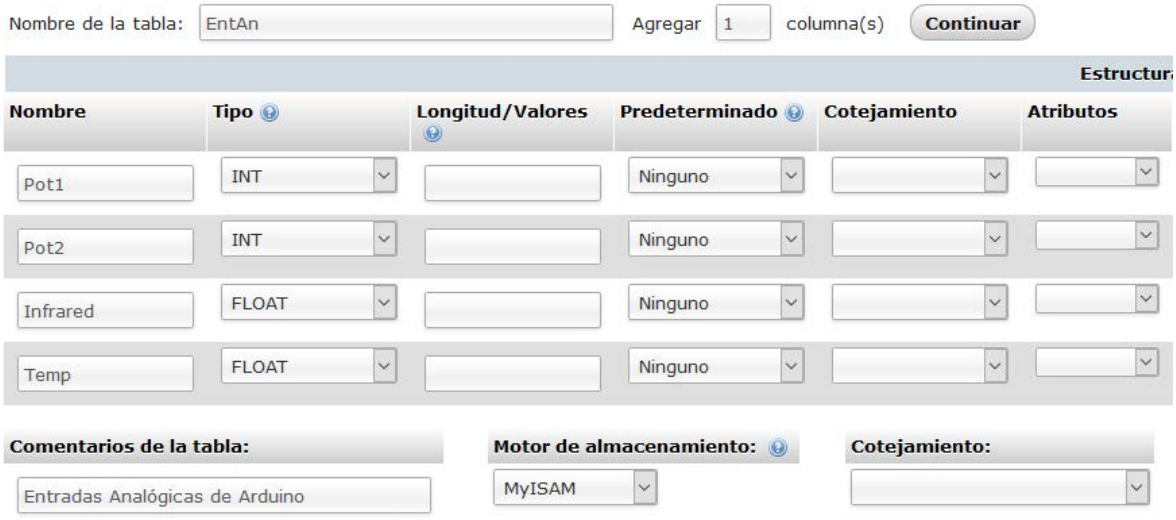

FIGURA 44

En la vista de "Estructura" podemos ver el resultado. Previamente, y desde esa misma vista vamos a crear una nueva columna. En la parte inferior de la vista tenemos la opción de agregar una columna. Añadiremos una al final de la tabla de nombre "Date" y tipo de dato "datetime". Tendremos la precaución de establecer que su valor predeterminado sea "CURRENT_TIMESTAMP". El motivo es tener una referencia de cuándo se han introducido los datos, a la par que una herramienta para ordenarlos correctamente en la tabla. La estructura creada la podemos ver en la Figura 45.

FIGURA 45

De forma análoga crearemos la tabla de entradas digitales con 4 columnas de tipo booleano, la tabla de salidas analógicas, con una columna de tipo entero y la tabla de salidas digitales con 4 columnas de tipo booleano. En todas ellas añadiremos la columna con la fecha y la hora de introducción.

Una vez creada la base de datos vamos a abordar el acceso a la misma mediante una interfaz web. Para ello vamos a crear una pequeña jerarquía de archivos en php y html para llevar a cabo este proceso. En primer lugar vamos a crear, mediante un editor de texto un archivo que denominaremos "connect.php" con el siguiente contenido:

```
<?php
$host = "mysql.hostinger.es";
$usuario = "u757591748_ardu";
$clave = "ArduinoYun";
$db = "u757591748_yun";
?>
```

Como vemos se trata de configurar todos los parámetros asociados a la base de datos que hemos creado. Para subirlo a nuestro alojamiento, utilizaremos la herramienta "Administrador de archivos 2" que aparece en el tablero bajo el icono  . Ingresaremos en la carpeta "public_html" y, mediante la opción "Upload" lo buscaremos en nuestro disco duro y lo subiremos.

De forma similar vamos a crear una "capa" de archivos php encargados de manejar el intercambio con la base de datos. Lo podríamos hacer con uno solo pero, por motivos de claridad y modularidad vamos a crear uno para cada tipo de señales manejadas. Estos serán:

- consultaEntDi.php
- consultaEntAn.php
- consultaSalDi.php
- consultaSalAn.php

Por encima de ellos tendremos la interfaz con el usuario, formada por otros tantos ficheros que interactuarán con el usuario y con los anteriores:

- entratasdigitales.php
- entradasanalogicas.php
- salidasdigitales.html
- salidasanalogicas.html

Por encima de todos ellos tendremos la página de inicio: index.html. En la Figura 46 podemos ver la arquitectura del conjunto, en la que es posible apreciar las diferencias entre trabajar con entradas y con salidas. Al trabajar con entradas de Arduino, la página principal conecta directamente con las páginas de consulta, que recaban la información de la base de datos antes de mostrársela al usuario. En el caso del acceso a salidas de Arduino, se pedirá en primer lugar al usuario que introduzca los valores que desea para ser enviados a la base de datos posteriormente. Vemos también que se han mezclado archivos php y html. No es una cuestión muy relevante, ya que es posible embeber código php en html y viceversa.

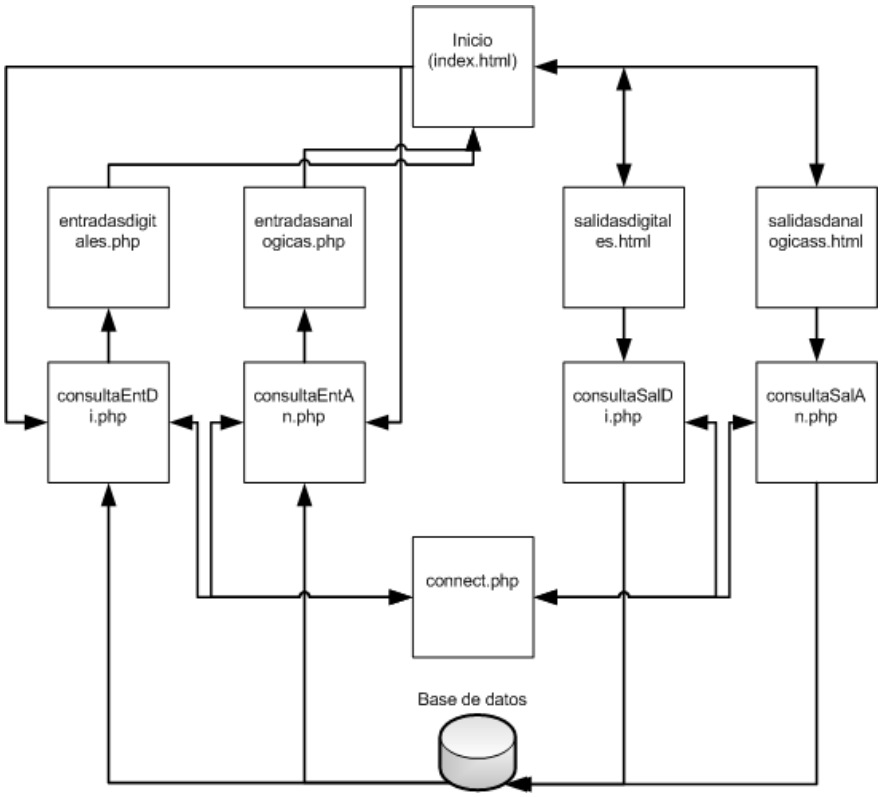

FIGURA 46

ARCHIVOS DE INTERFAZ DE USUARIO

Nos estamos refiriendo a las dos filas superiores de la Figura 46. Estos archivos van a ser presentados al usuario para su interacción con la base de datos y, eventualmente, con Arduino. No obstante, en este apartado completaremos la explicación con el contenido de los ficheros que se encuentran por debajo de los señalados.

El primero de ellos es index.html. Se trata de una página de entrada en la que el usuario va a seleccionar el tipo de señales con las que quiere trabajar.

```
<html>
<head>
Seleccione el tipo de señales con las que quiere trabajar:
</head>
<body>
<form action="salidasdigitales.html"><input type="submit" value="Salidas Digitales" /></form>
<form action="salidasanalogicas.html"><input type="submit" value="Salidas Analógicas" /></form>
<form action="consultaEntDi.php"><input type="submit" value="Entradas Digitales" /></form>
<form action="consultaEntAn.php"><input type="submit" value="Entradas Analógicas" /></form>
</body>
</html>
```

El formato es el más sencillo posible. Por razones de claridad hemos relegado completamente la estética, pero se anima a quien pueda estar interesado a que explore formas más elegantes

de presentar las distintas páginas. Index nos presenta un formulario con 4 botones que nos permiten acceder a las páginas de trabajo con las señales requeridas. Al pulsar el botón correspondiente se ejecuta la acción de llamada al fichero asociado. A continuación vemos el contenido del fichero "salidasdigitales.html:

```
<html>
<body>
<form action="consultaSalDi.php" method="post">
   Marca las salidas que quieras activar<br>
D6: <input type="checkbox" name="D6" value="Yes" /><br>
D9: <input type="checkbox" name="D9" value="Yes" /><br>
D10: <input type="checkbox" name="D10" value="Yes" /><br>
D11: <input type="checkbox" name="D11" value="Yes" /><br>
   <input type="submit" name="formSubmit" value="Enviar" />
</form>
</body>
</html>
```

De nuevo se trata de un formulario, en este caso con entradas del tipo "checkbox". Al tratarse de señales digitales, aquellas seleccionadas por el usuario se remitirán a la base de datos como activas y el resto no. El botón de enviar es el que ejecuta la acción de llamada al fichero "consultaSalDi.php" y le envía el estado de las 4 señales mediante el método "post". Este fichero va a recoger la información testeando qué señales le han sido enviadas. En este caso su valor es irrelevante ya que el hecho de haber sido seleccionadas indica que el usuario desea activarlas a diferencia del resto:

```
<?php
include ("connect.php");
$tabla = SalDi;

if(isset($_POST['D6'])) $D6 = 1;
else $D6 = 0;
if(isset($_POST['D9'])) $D9 = 1;
else $D9 = 0;
if(isset($_POST['D10']))$D10 = 1;
else $D10 = 0;
if(isset($_POST['D11']))$D11 = 1;
else $D11 = 0;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");
mysqli_query($conexion, "INSERT INTO $tabla (D6,D9,D10,D11) VALUES ($D6,$D9,$D10,$D11)") or
die ("Error en la base de datos");

echo 'Dato almacenado';
mysqli_close($conexion);
?>
<html>
<form action="index.html"><input type="submit" value="Volver" /></form>
</html>
```

El proceso comienza con la inclusión del fichero "connect.php" que contiene los parámetros necesarios para conectar con la base de datos. A continuación se comprueba qué señales se han recibido mediante el método "post". Éstas se pondrán a 1 quedando el resto a 0. Después

de conectar con la base de datos se insertan en la tabla “SalDi” tal y como se indica en la variable correspondiente. Finalmente se inserta un formulario html consistente en un botón devuelta al índice, al tiempo que se indica que la información ha sido almacenada en la base de datos.

La gestión de la salida analógica funciona de una forma similar. El fichero de interfaz “salidasanalogicas.html” permite al usuario introducir el valor deseado para la señal, en este caso la intensidad del diodo led conectado a la salida 6 de arduino.

```
<html>
<body>
<form action="consultaSalAn.php" method="post">
   Introduce el valor de intensidad del Led blanco (0-255)<br>
Led: <input type="number" name="Led" min="0" max="255" step="1" value="30">
   <input type="submit" name="formSubmit" value="Enviar" />
</form>
</body>
</html>
```

De nuevo se trata de un formulario. En este caso tiene una sola entrada de tipo “number” cuyos valores permitidos están entre 0 y 255 (los admitidos por arduino para este tipo de señales PWM). El valor por defecto es 30. Al pulsar el botón “Enviar”, de nuevo el método “post” trasladará a “consultaSalAn.php” el valor seleccionado.

```
<?php
include ("connect.php");
$tabla = SalAn;

if(isset($_POST['Led']) && !empty($_POST['Led']))
{
$LedBlanco = $_POST['Led'];
$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");
mysqli_query($conexion, "INSERT INTO $tabla (Led) VALUES ($LedBlanco)") or die ("Error en la base
de datos");

echo 'Dato almacenado';
}
else echo 'Entrada incorrecta';
mysqli_close($conexion);
?>
<html>
<form action="index.html"><input type="submit" value="Volver" /></form>
</html>
```

El funcionamiento es análogo al caso anterior. La tabla seleccionada lógicamente es en este caso SalAn y, a diferencia del caso de las salidas digitales, el parámetro valor sí tiene significado. De hecho se añade una comprobación de que no se haya dejado en blanco, en cuyo caso se retorna el mensaje “Entrada incorrecta”. Si todo es correcto, el valor se envía a la base de datos.

En el caso de las entradas, la Figura 46 muestra que “index.html” no llama a los ficheros de interfaz sino a los de consulta, ya que será necesario leer los valores de la base de datos antes de mostrarlos al usuario. Las entradas digitales se consultan en “consultaEntDi.php”.

```
<?php
include ("connect.php");
$tabla = EntDi;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");
$resultado = mysqli_query($conexion, "SELECT D4,D7,D8,D12 FROM $tabla") or die ("Error en la base de datos");

while($row = mysqli_fetch_assoc($resultado))
{
$D4 = $row['D4'];
$D7 = $row['D7'];
$D8 = $row['D8'];
$D12 = $row['D12'];
}

mysqli_close($conexion);
?>
<html>
<form action="entradasdigitales.php" method="post" name="entradas">
<input type="hidden" name="D4" value="<?php echo $D4; ?>" >
<input type="hidden" name="D7" value="<?php echo $D7; ?>" >
<input type="hidden" name="D8" value="<?php echo $D8; ?>" >
<input type="hidden" name="D12" value="<?php echo $D12; ?>" >
</form>
<script type="text/javascript">document.entradas.submit();</script>
</html>
```

La tabla que se va a consultar en este caso es “EntDi”. De ella se recogen en la variable “$resultado” los valores contenidos en la tabla. La función “mysqli_fetch_assoc” fundamentalmente los asocia por filas a un array (“$row”) que contiene los valores de cada columna. El bucle “while” va recorriendo todas las filas y asociando los valores encontrados a las variables “$D4”, “$D7”, “$D8” y “$D12”. De la forma en que está presentado, los valores se van sobrescribiendo en cada iteración, por lo que los valores finales corresponderán con la última fila de la tabla. Si hemos ordenado la tabla por fecha en sentido ascendente a través del menú de “Operaciones” de la propia tabla, nos estaremos quedando con el valor más actual. Este procedimiento es antieconómico, ya que se recorre toda la tabla para quedarse solamente con el último valor. Sería más eficiente ordenar en sentido descendente y quedarse con el primero, pero se ha dejado así con el fin de reflejar cómo se puede acceder a toda la información.

Los valores obtenidos de la base de datos se pasan a “entradasdigitales.php” para ser presentados al usuario. Se utiliza para ello un formulario html como hasta ahora, pero con las entradas ocultas (“hidden”), ya que el usuario no tiene intervención en el proceso.

```
<html>
<?php
if(isset($_POST['D4'])) $D4 = $_POST['D4'];
```

```
if(isset($_POST['D7'])) $D7 = $_POST['D7'];
if(isset($_POST['D8'])) $D8 = $_POST['D8'];
if(isset($_POST['D12'])) $D12 = $_POST['D12'];
?>

Estado actual de las señales:
<table style="width:100%" >
<table border='1'>
<tr>
<th>D4</th>
<th>D7</th>
<th>D8</th>
<th>D12</th>
</tr>
<tr>
<td><?php echo $D4; ?></td>
<td><?php echo $D7; ?></td>
<td><?php echo $D8; ?></td>
<td><?php echo $D12; ?></td>
</tr>
</table>
<form action="consultaEntDi.php"><input type="submit" value="Actualizar" /></form>
<form action="index.html"><input type="submit" value="Volver" /></form>
</html>
```

El estado de las señales se recoge mediante el método “post” y se presenta al usuario a través de una sencilla tabla escrita en html. Se añade un botón “Actualizar” para volver a leer la base de datos en cualquier momento y un botón “Volver” para regresar al menú principal.

De forma análoga se trabaja con las entradas analógicas. El fichero “consultaEntAn.php” és llamado por “index.html”, recoge la información de la base de datos y se la envía a “entradasanalogicas.php” mediante el método “post”.

```
<?php
include ("connect.php");
$tabla = EntAn;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");
$resultado = mysqli_query($conexion, "SELECT Pot1,Pot2,Infrared,Temp FROM $tabla") or die ("Error
en la base de datos");

while($row = mysqli_fetch_assoc($resultado))
{
$Pot1 = $row['Pot1'];
$Pot2 = $row['Pot2'];
$Infrared = $row['Infrared'];
$Temp = $row['Temp'];
}

mysqli_close($conexion);
?>
<html>
<form action="entradasanalogicas.php" method="post" name="entradas">
```

```
<input type="hidden" name="Pot1" value="<?php echo $Pot1; ?>" >
<input type="hidden" name="Pot2" value="<?php echo $Pot2; ?>" >
<input type="hidden" name="Infrared" value="<?php echo $Infrared; ?>" >
<input type="hidden" name="Temp" value="<?php echo $Temp; ?>" >
</form>
<script type="text/javascript">document.entradas.submit();</script>
</html>
```

La interfaz "entradasanalogicas.php" vuelve a comportarse de forma análoga al caso de las entradas digitales:

```
<html>
<?php
if(isset($_POST['Pot1'])) $Pot1 = $_POST['Pot1'];
if(isset($_POST['Pot2'])) $Pot2 = $_POST['Pot2'];
if(isset($_POST['Infrared'])) $Infrared = $_POST['Infrared'];
if(isset($_POST['Temp'])) $Temp = $_POST['Temp'];
?>

Estado actual de las señales:
<table style="width:100%" >
<table border='1'>
<tr>
<th>Pot1</th>
<th>Pot2</th>
<th>Infrared</th>
<th>Temp</th>
</tr>
<tr>
<td><?php echo $Pot1; ?></td>
<td><?php echo $Pot2; ?></td>
<td><?php echo $Infrared; ?></td>
<td><?php echo $Temp; ?></td>
</tr>
</table>
<form action="consultaEntAn.php"><input type="submit" value="Actualizar" /></form>
<form action="index.html"><input type="submit" value="Volver" /></form>
</html>
```

ARCHIVOS DE INTERFAZ CON ARDUINO

La Figura 46 nos mostraba la arquitectura software de la interfaz del usuario externo con la base de datos. De cara a la conexión con arduino, en nuestro alojamiento externo vamos a tener que prever una cierta infraestructura software. La Figura 47 nos muestra el esquema general de la misma.

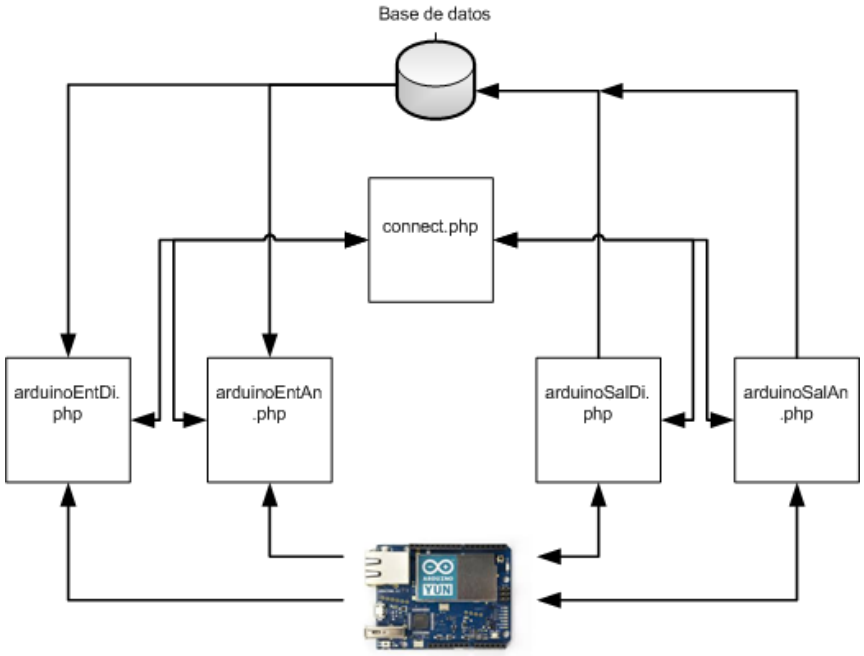

FIGURA 47

Arduino envía el estado de sus entradas a los módulos de gestión correspondientes. En el caso de las salidas, la comunicación es bidireccional ya que la iniciativa de su consulta parte de arduino, que solicita conocer el estado que requiere el usuario para las salidas. Los módulos de gestión de responden proporcionando la información requerida. Siguiendo el mismo orden que en el apartado anterior, empezamos explicando el funcionamiento del módulo de gestión de entradas digitales.

```
<?php
include ("connect.php");
$tabla = EntDi;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");

if(isset($_POST['D4'])) $D4 = $_POST['D4'];
else $D4 = 0;
if(isset($_POST['D7'])) $D7 = $_POST['D7'];
else $D7 = 0;
if(isset($_POST['D8']))$D8 = $_POST['D8'];
else $D8 = 0;
if(isset($_POST['D12']))$D12 = $_POST['D12'];
else $D12 = 0;

mysqli_query($conexion, "INSERT INTO $tabla (D4,D7,D8,D12) VALUES ($D4,$D7,$D8,$D12)") or die
("Error en la base de datos");
mysqli_close($conexion);
?>
```

Vemos que tiene una estructura análoga a la que tenían los módulos de la parte de la interfaz de usuario. En este caso en lugar de provenir la información de un “post” de otro módulo en el

servidor, proviene de un software en arduino. Análogo es también el funcionamiento del módulo de gestión de entradas analógicas:

```
<?php
include ("connect.php");
$tabla = EntAn;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");

if(isset($_POST['Pot1'])) $Pot1 = $_POST['Pot1'];
else $Pot1 = 0;
if(isset($_POST['Pot2'])) $Pot2 = $_POST['Pot2'];
else $Pot2 = 0;
if(isset($_POST['Infrared']))$Infrared = $_POST['Infrared'];
else $Infrared = 0;
if(isset($_POST['Temp']))$Temp = $_POST['Temp'];
else $Temp = 0;

mysqli_query($conexion, "INSERT INTO $tabla (Pot1,Pot2,Infrared,Temp) VALUES ($Pot1,$Pot2,$Infrared,$Temp)") or die ("Error en la base de datos");
mysqli_close($conexion);
?>
```

El trabajo con las señales de salida tampoco presenta muchas diferencias en este ámbito.

Una vez que tenemos la base de datos y el sitio web para acceder a ella, vamos a crear una nueva infraestructura para soportar los intercambios de datos con Arduino. De nuevo con el fin de tener la máxima claridad y modularidad, vamos a crear un fichero por cada tipo de señal:

- arduinoEntDi.php
- arduinoEntAn.php
- arduinoSalDi.php
- arduinoSalAn.php

La relación de estos módulos de gestión con la base de datos la podemos observar en la Figura 48.

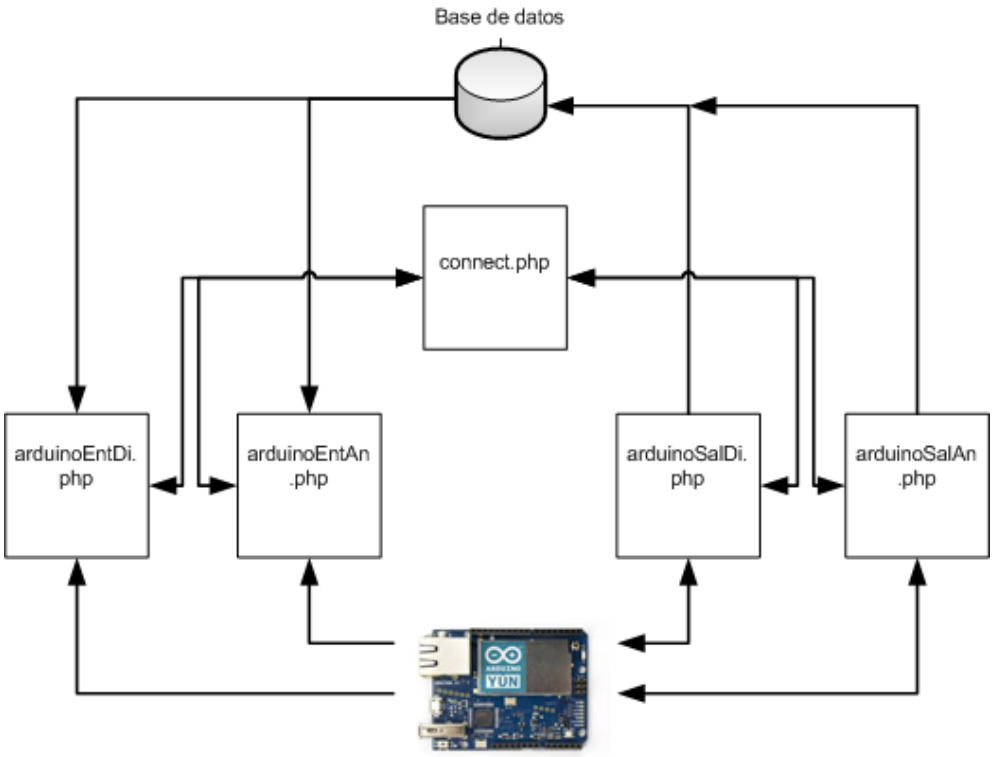

FIGURA 48

Se puede distinguir que los módulos que gestionan las entradas reciben un flujo de datos únicamente entrante desde arduino, mientras que en los que gestionan salidas, el flujo no es solo saliente sino bidireccional. Esto se debe a que en este modelo de comunicación la iniciativa siempre parte de arduino. En el caso de las señales salientes el envío de la información se produce tras la petición de la misma.

Veamos en primer lugar la forma de tratar las entradas digitales:

```
<?php
```

```
include ("connect.php");
$tabla = EntDi;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");

if(isset($_POST['D4'])) $D4 = $_POST['D4'];
else $D4 = 0;
if(isset($_POST['D7'])) $D7 = $_POST['D7'];
else $D7 = 0;
if(isset($_POST['D8']))$D8 = $_POST['D8'];
else $D8 = 0;
if(isset($_POST['D12']))$D12 = $_POST['D12'];
else $D12 = 0;

mysqli_query($conexion, "INSERT INTO $tabla (D4,D7,D8,D12) VALUES ($D4,$D7,$D8,$D12)") or die
("Error en la base de datos");
mysqli_close($conexion);
?>
```

Este módulo abre una conexión con la base de datos y comprueba si le han sido remitidas mediante el método "POST" cada una de las señales, en cuyo caso actualiza el estado de la variable correspondiente con el contenido del envío. Una vez recogidas todas, se envía a la base de datos y se cierra la conexión.

El trabajo con las entradas analógicas es totalmente análogo:

```
<?php
include ("connect.php");
$tabla = EntAn;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");

if(isset($_POST['Pot1'])) $Pot1 = $_POST['Pot1'];
else $Pot1 = 0;
if(isset($_POST['Pot2'])) $Pot2 = $_POST['Pot2'];
else $Pot2 = 0;
if(isset($_POST['Infrared']))$Infrared = $_POST['Infrared'];
else $Infrared = 0;
if(isset($_POST['Temp']))$Temp = $_POST['Temp'];
else $Temp = 0;

mysqli_query($conexion, "INSERT INTO $tabla (Pot1,Pot2,Infrared,Temp) VALUES
($Pot1,$Pot2,$Infrared,$Temp)") or die ("Error en la base de datos");
mysqli_close($conexion);
?>
```

Las salidas tampoco requieren de un método sustancialmente distinto:

```
<?php
include ("connect.php");
$tabla = SalDi;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");
$resultado = mysqli_query($conexion, "SELECT D6,D9,D10,D11 FROM $tabla") or die ("Error en la base
```

```
de datos");

while($row = mysqli_fetch_assoc($resultado))
{
$D6 = $row['D6'];
$D9 = $row['D9'];
$D10 = $row['D10'];
$D11 = $row['D11'];
}
echo "D6=$D6&D9=$D9&D10=$D10&D11=$D11"; //Enviamos el último valor presente de cada una
mysqli_close($conexion);
?>
```

El hecho de que se active el módulo implica que arduino ha requerido conocer el estado de las salidas que él gestiona, por lo que solo tiene que conectarse a la base de datos, leerlas y enviarlas. En el ejemplo de las salidas digitales vemos cómo se pueden ir recorriendo las distintas filas de la tabla que alberga el estado de las mismas. A medida que se van leyendo se van sobreescribiendo los resultados, por lo que simplemente nos vamos a quedar con el último. Esto no es muy eficiente en realidad, ya que sería mejor ordenar la tabla en sentido contrario y leer únicamente el primero. A pesar de ello lo hemos mantenido de este modo con el fin de que sirva como ejemplo para ver cómo recorrer la tabla completa.

La forma de devolver los valores es mediante un mensaje en el que se unen todos ellos mediante símbolos “&”. Se podría haber hecho de otra forma; basrta con que arduino conozca el formato para poder extraer de él la información relevante. Para enviarlo se emplea el comando “echo” cuya salida se redirige automáticamente hacia arduino, ya que ha sido él quien ha invocado el procedimiento.

De forma análoga se trabaja con la única salida analógica en este caso:

```
<?php
include ("connect.php");
$tabla = SalAn;

$conexion = mysqli_connect ($host, $usuario, $clave, $db) or die ("No se encuentra el servidor");
$resultado = mysqli_query($conexion, "SELECT Led FROM $tabla") or die ("Error en la base de datos");

while($row = mysqli_fetch_assoc($resultado))
{
  $Led = $row['Led'];
}
echo "Led=$Led"; //Enviamos el último valor presente
mysqli_close($conexion);
?>
```

Para completar el proceso necesitamos programar Arduino Yun para que interactúe con las señales reales por un lado y se comunique con el servidor por el otro. Vamos a construir un programa modular en el que la gestión de cada tipo de señales irá separada del resto en una función propia.

En primer lugar vamos a examinar el programa principal:

```
#include <Bridge.h>
#include <YunClient.h>
//#include <YunServer.h>

YunClient client;
const char *server = "www.checam.esy.es"; // URL del servidor Hostinger
String datos ="";          //String para enviar datos a Hostinger

void setup() {
Bridge.begin();
Serial.begin(9600);
pinMode(13,OUTPUT);
digitalWrite(13, LOW);
//Salidas digitales en la shield
pinMode(10, OUTPUT);
pinMode(11, OUTPUT);
pinMode(9, OUTPUT);
pinMode(6, OUTPUT);
//Entradas digitales en la shield
pinMode(12, INPUT);
pinMode(8, INPUT);
pinMode(7, INPUT);
pinMode(4, INPUT);
}
```

Vemos en primer lugar las declaraciones y archivos de cabecera que vamos a necesitar. A continuación, la función "setup()" inicia la comunicación hacia el PC y hacia el procesador Atheros, además de establecer la naturaleza de las señales digitales. Además de las señales de la "shield" vamos a recurrir a la salida 13, conectada en la placa de arduino a un Led de color rojo para tareas de monitorización del estado de las conexiones.

Dentro de "loop()" vamos a secuenciar los diferentes eventos de comunicación asociados a cada tipo de señal. Introduciremos retardos entre cada par de operaciones por dos motivos:

- Para dar tiempo de respuesta al servidor.
- Porque no tiene especial interés alcanzar las máximas tasas de intercambio de información posibles. En este sentido se puede optar por la transmisión a intervalos regulares o por realizarla cuando se produzca un determinado evento. También es importante señalar que la utilización del retardo "delay()" no es la opción más interesante. Sería preferible por motivos de consumo incluir alguna librería que nos permita acceder a los modos de bajo consumo de arduino. No lo abordamos aquí para no desviar la atención del tema que estamos tratando.

```
void loop() {
 enviarEntDi();
  delay(2000);
 enviarEntAn();
  delay(2000);
 if (enviarSalAn()){
  delay(2000);
  recibirSalAn();
 }
  delay(2000);
```

```
  if (enviarSalDi()){
   delay(2000);
   recibirSalDi();
  }
}
```

Se aprecia cómo el estado de las entradas se envía sin más, mientras que las salidas requieren de dos fases: envío de la consulta y recepción de la respuesta. Esta última operación se efectúa si la consulta se ha completado con éxito.

Si examinamos el código correspondiente al envío del estado de las entradas digitales (enviarEntDi()), vemos que el proceso arranca con la conexión al servidor, sigue con el envío de una cabecera y finaliza con el envío de la información. Previamente se ha construido la cadena de datos incluyendo el estado de las señales:

```
void enviarEntDi(){
   if (client.connect(server, 80)) {
   Serial.println("connectado");
   digitalWrite(13, LOW);
   delay(4000);
   datos="D4="+String(digitalRead(4)) + "&D7="+
String(digitalRead(7))+"&D8="+String(digitalRead(8))+"&D12="+String(digitalRead(12));
   client.println("POST /arduinoEntDi.php HTTP/1.1");
   client.print("Host: ");
   client.println(server);
   client.println("User-Agent: Arduino/1.0");
   client.println("Content-Type: application/x-www-form-urlencoded;");
   client.print("Content-Length: ");
   client.println(datos.length());
   client.println();
   client.println(datos);
   client.println("Connection: close");
   Serial.println(datos);
  }
  else{
   Serial.println("Error de conexión");
   digitalWrite(13,HIGH);
   delay(3000);
  }
  if(client.connected()){
   client.stop();   //desconectar después del envío
   Serial.println("desconnectado");
  }
 }
```

Es muy importante la sintaxis de la cabecera, ya que de no respetarse estrictamente los campos y el formato, el servidor no será capaz de interpretarla y el intercambio de información no se producirá, no retornando ningún código de error que nos permita rastrear el problema.

El envío de las señales analógicas es exactamente el mismo. Simplemente cambia la construcción de la cadena de contenido:

```
  datos="Pot1="+String(analogRead(0)) + "&Pot2="+
String(analogRead(1))+"&Infrared="+String(analogRead(3))+"&Temp="+String(analogRead(4));
```

La recepción del estado de las salidas ya hemos comentado que es un proceso doble, por ese motivo se ha desdoblado en dos procedimientos para cada tipo de señales. Vemos en primer lugar el caso de las salidas digitales:

enviarSalDi()

```
bool enviarSalDi(){
  bool exito = false;
   if (client.connect(server, 80)) {
  Serial.println("connectadoSalDi");
  digitalWrite(13, LOW);
  exito = true;
  delay(4000);

  client.println("POST
/arduinoSalDi.php HTTP/1.1");
  client.print("Host: ");
  client.println(server);
  client.println("User-Agent:
Arduino/1.0");
  client.println("Content-Type:
application/x-www-form-
urlencoded;");
  client.println("Connection: close");
  client.println();
 }
 else{
  Serial.println("Error de conexión");
  digitalWrite(13,HIGH);
  delay(3000);
 }
 return (exito);
}
```

recibirSalDi()

```
void recibirSalDi(){
 String respuesta = "", dato = "";
 char c;
 int indice1,indice2,valor;
 while (client.available()>0) {
  c = client.read();
  respuesta = respuesta + c;
 }
 Serial.print("Mensaje: ");  // contiene el mensaje completo
 Serial.println(respuesta);
 indice1= respuesta.indexOf('=')+1;
 indice2= respuesta.indexOf('&')-1;
 dato = respuesta.charAt(indice1);
 Serial.print("D6: ");
 Serial.println(dato);
 valor = dato.toInt();
 digitalWrite(6,valor);
 respuesta = respuesta.substring(indice2+2);
 Serial.print("Mensaje: ");
 Serial.println(respuesta);
 indice1= respuesta.indexOf('=')+1;
 indice2= respuesta.indexOf('&')-1;
 dato = respuesta.charAt(indice1);
 Serial.print("D9: ");
 Serial.println(dato);
 valor=dato.toInt();
 digitalWrite(9, valor);
 respuesta = respuesta.substring(indice2+2);
 Serial.print("Mensaje: ");
 Serial.println(respuesta);
 indice1= respuesta.indexOf('=')+1;
 indice2= respuesta.indexOf('&')-1;
 dato = respuesta.charAt(indice1);
 Serial.print("D10: ");
 Serial.println(dato);
 valor=dato.toInt();
 digitalWrite(10, valor);
 respuesta = respuesta.substring(indice2+2);
 Serial.print("Mensaje: ");
 Serial.println(respuesta);
 indice1= respuesta.indexOf('=')+1;
 dato = respuesta.charAt(indice1);
 Serial.print("D11: ");
 Serial.println(dato);
 valor=dato.toInt();
 digitalWrite(11, valor);
```

```
  if(client.connected()){
    client.stop();   //desconectar
    Serial.println("desconnectadoSalAn");
   }
}
```

La estructura de la función de envío es sencilla, ya que se trata de enviaar únicamente una cabecera al módulo php correspondiente: arduinoSalDi en este caso. La función de recepción por el contrario, debe extraer de la cadena de respuesta la información relevante. El proceso programado consiste en ir recortando dicha cadena a medida que se van leyendo los datos. Para ello se recurre a los símbolos "=" y "&" introducidos en el lado del servidor, para localizar en qué posiciones de la cadena se encuentran los datos. Se han introducido impresiones por el puerto serie que serían prescindibles en funcionamiento normal, pero que nos permiten ver en tiempo real a través del monitor serie del IDE de arduino cómo se van extrayendo los datos:

```
connectadoSalDi
Mensaje: HTTP/1.1 200 OK
Date: Thu, 12 Jan 2017 14:21:52 GMT
Server: Apache
X-Powered-By: PHP/5.5.35
Content-Length: 21
Connection: close
Content-Type: text/html

D6=0&D9=1&D10=0&D11=1
D6: 0
Mensaje: D9=1&D10=0&D11=1
D9: 1
Mensaje: D10=0&D11=1
D10: 0
Mensaje: D11=1
D11: 1
```

El caso de la salida analógica resulta análogo, aunque más simple en el tratamiento de la respuesta, al tratarse de una sola señal:

enviarSalAn()	**recibirSalAn()**

bool enviarSalAn(){
  bool exito = false;
  if (client.connect(server, 80)) {
  Serial.println("connectadoSalAn");
  digitalWrite(13, LOW);
  exito = true;
  delay(4000);
 	
  client.println("POST /arduinoSalAn.php HTTP/1.1");
  client.print("Host: ");
  client.println(server);
  client.println("User-Agent: Arduino/1.0");
	void recibirSalAn(){
	  String respuesta = "";
	  char c;
	  int indice1,valor;
	  while (client.available()>0) {
	  c = client.read();
	  respuesta = respuesta + c;
	  }
	  Serial.println(respuesta); // contiene el mensaje completo
	  indice1= respuesta.indexOf('=')+1;
	  respuesta = respuesta.substring(indice1);
	  Serial.println(respuesta);
	  valor=respuesta.toInt();
	  analogWrite(6, valor);

```
    client.println("Content-Type:
application/x-www-form-
urlencoded;");
    client.println("Connection: close");
    client.println();
   }
   else{
    Serial.println("Error de conexión");
    digitalWrite(13,HIGH);
    delay(3000);
   }
   return (exito);
  }
```

```
  if(client.connected()){
    client.stop();   //desconectar
    Serial.println("desconnectadoSalDi");
   }
}
```

CONEXIÓN DE ARDUINOS EN CASCADA

A pesar de la amplia gama de placas Arduino disponibles y de que alguna de ellas proporciona notables capacidades de conexión y procesamiento, cuando la aplicación presenta una serie de demandas a nivel de hardware y/o software, es posible que no encontremos una placa que cumpla con todos los requerimientos. Esto puede deberse a varios motivos:

- Exceso de señales de algún tipo concreto.
- Exceso de señales a manejar en conjunto.
- Funcionalidades adicionales a soportar: almacenamiento, conectividad, etc.
- Capacidad del procesador para responder en tiempo real a las demandas de la aplicación.
- Conflictos en el uso de interrupciones.

Cuando se dan estas circunstancias en alguna medida una solución puede ser repartir el trabajo entre varios Arduinos. Esto relaja las exigencias para cada uno de ellos pero impone dos demandas adicionales:

- Comunicación: para poder intercambiar/centralizar la información que manejan los Arduinos de forma distribuida.
- Sincronización: para que la adquisición de señales y control de dispositivos se pueda llevar a cabo de manera coordinada.

Existen múltiples opciones que nos podemos plantear, tanto para la comunicación como para la sincronización, que se pueden implementar por separado o de manera conjunta. Vamos a centrarnos en este caso en la opción que puede ser más intuitiva: la comunicación serie.

A través del puerto o puertos serie podemos conseguir que nuestros Arduinos se comuniquen entre sí y con el PC. De esta forma se pueden intercambiar el estado de la periferia conectada a cada uno y enviarse órdenes. Estos dos aspectos conforman los requisitos básicos para disponer de comunicación y sincronización. En el caso de la sincronización esta opción podría quedarse un tanto corta en algunas aplicaciones, por lo que sería necesario implementarla de forma independiente a la comunicación utilizando por ejemplo señales digitales para "disparar" eventos en otro Arduino.

Independientemente de la disponibilidad de puertos serie de cada modelo de Arduino, la librería SoftwareSerial permite habilitar nuevos puertos, aunque con ciertas limitaciones que dependen de cada placa.

Esta opción la vamos a emplear en Arduino UNO para retransmitir la información a Arduino Yun a través de las patillas 10(Rx) y 11(Tx). En Arduinos con más de un puerto serie, como Mega podemos emplear la clase "serial1" y siguientes, las cuales funcionan de forma análoga a la clase "serial" habitual. El puerto serie 1 en Mega corresponde a las patillas 18(Tx1) y 19(Rx1). Yun dispone también del puerto serie 1 en las patillas 0 y 1 pero son utilizadas por la librería "Bridge" para comunicar con el procesador Atheros, por lo que ambas funciones son incompatibles. En la Figura 49 vemos cómo sería la conexión entre ambos dispositivos.

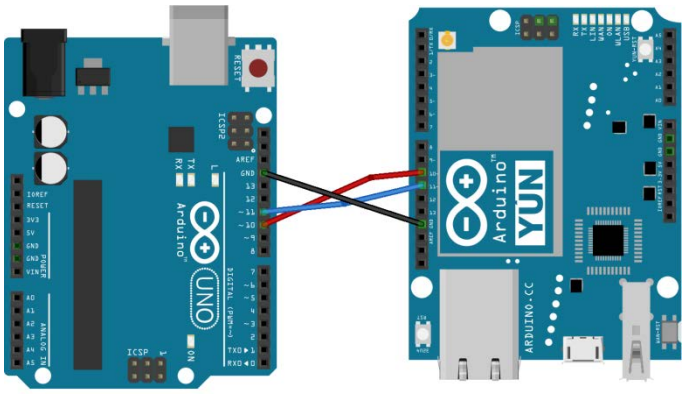

FIGURA 49

Las líneas de transmisión y recepción se cruzan: Tx-Rx, Rx-Tx y se une la masa (GND) de ambos para que trabajen sobre una referencia común.

De esta forma el conjunto suma la periferia de ambos dispositivos y dispone de la capacidad de almacenamiento y conectividad de YUN.

Un ejemplo muy sencillo para comprobar cómo las órdenes generadas desde la interfaz de PC se pueden propagar a través de Arduino UNO a YUN consiste en tomar la orden de encendido/apagado de una salida digital (por ejemplo la 11) y emplearla para encender el diodo led integrado (salida 13) que incorporan ambas placas.

Arduino UNO	**Arduino YUN**

#include <SoftwareSerial.h>
SoftwareSerial mySerial(10, 11); // RX, TX
String palabra;
void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(115200);
  while (!Serial) {;}
  mySerial.begin(4800);
}
 	
void loop() {
  if (Serial.available()) {
  palabra=Serial.readString();
  if(palabra == "D11ON\n")
  digitalWrite(13, HIGH);
  else if(palabra == "D11OFF\n")
  digitalWrite(13, LOW);
  mySerial.print(palabra);
  }
}
	#include <SoftwareSerial.h>
	SoftwareSerial mySerial(10, 11); // RX, TX
	String palabra;
	void setup() {
	pinMode(13, OUTPUT);
	mySerial.begin(4800);
	}
	
	void loop() {
	  if (mySerial.available()) {
	  digitalWrite(13, LOW);
	  palabra=mySerial.readString();
	  if(palabra == "D11ON\n")
	  digitalWrite(13, HIGH);
	  else if(palabra == "D11OFF\n")
	  digitalWrite(13, LOW);
	  }
	}

REFERENCIAS

Paseo guiado por Visual C++

Getting started with Windows forms

Crear una aplicación de Windows Forms

How mouse input Works on Windows Forms

Librería de comunicación serie con Arduino

Guía de Microsoft para comunicaciones por puerto serie en Windows

Conexión de arduino con alojamiento en Hostinger (vídeos de youtube)