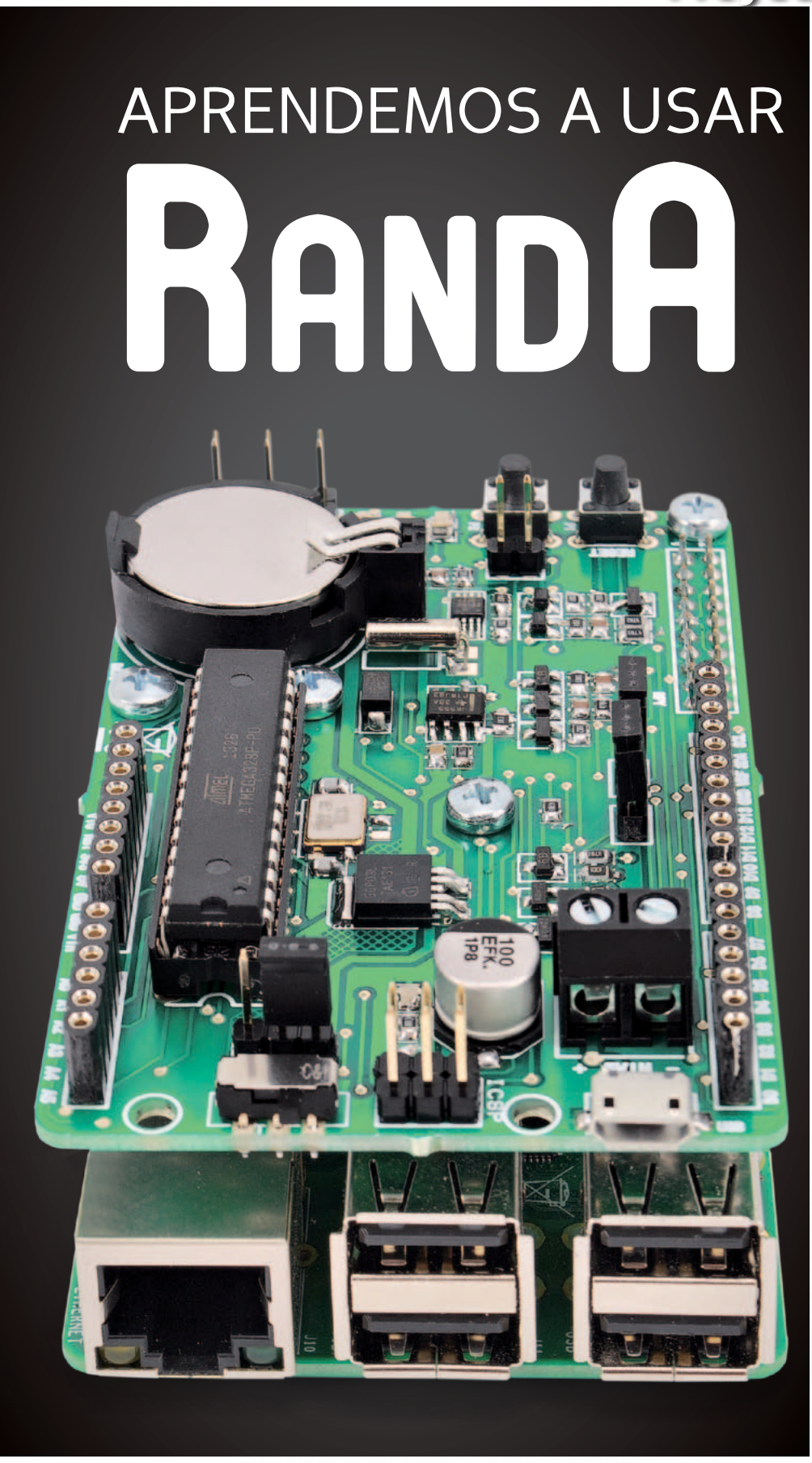
APRENDEMOS A USAR RandA



Ha llegado el momento de utilizar la tarjeta que une el potencial de Raspberry Pi al sencillo uso de Arduino.

DANIELE DENARO

Los mundos de Raspberry Pi y Arduino se han tocado, pero sin colisión: al contrario, ha surgido una cooperación que ha dado a luz un hibrido de enorme potencial. Hemos empezado a hablar en la edición de Febrero de 2015, presentando la tarjeta RandA (este nombre significa **R**aspberry **and** **A**rduino) o una tarjeta de prototipo dotado de un "core" Arduino, pero provista tanto de los clásicos conectores hembra para conectar los shield de Arduino, como de las conexiones para conectarse al conector de expansión de Raspberry Pi (también de la versión ultima nacida de la "frambuesa").
En el artículo de la revista 324 hemos visto las características principales de RandA y ahora profundizamos algunos aspectos con ejemplos concretos de uso.
Damos por descontado el éxito en la instalación del paquete software de RandA, que comprende el IDE Arduino modificado a utilizar en el propio ordenador y de todo el software previsto en Raspberry Pi; para la instalación, remitimos al artículo introductorio y los archivos README presentes en la distribución. Recordamos que quien esté interesado, podrá adquirir la tarjeta SD para Raspberry Pi con la instalación ya lista.
En **Fig. 1** resumimos el esquema de RandA y de los puentes.
Como hemos visto, la programa-

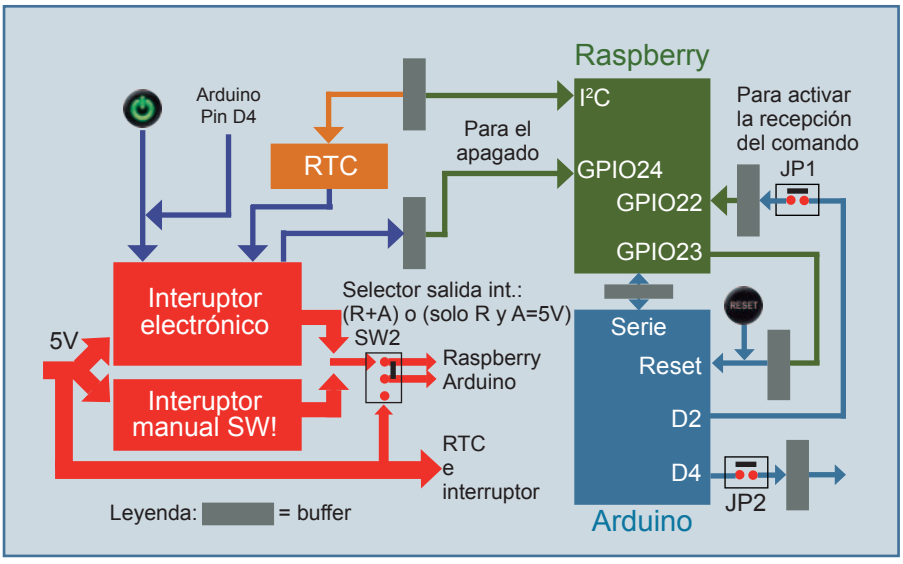


***Fig. 1** - Hardware de RandA.*

ción y el uso de la tarjeta RandA prevé dos tipos de enfoque: uno Arduino-céntrico, donde es Arduino el que "guía los bailes", y un enfoque Raspberry-céntrico, donde sin embargo el ambiente de referencia es el sistema operativo Linux.

ENFOQUE ARDUINO-CÉNTRICO

Partimos del enfoque Arduino-céntrico y comenzamos a desarrollar un sketch sobre nuestro ordenador utilizando el IDE Arduino modificado. El IDE resulta perfectamente compatible con el uso típico de una tarjeta Arduino conectada localmente al puerto USB de nuestro ordenador, por tanto, para quien ha utilizado ya este entorno de programación no hay nada más que añadir.

La sorpresa llega en el momento de seleccionar, entre aquellos disponibles, el puerto al cual Arduino está conectado. Junto a los puertos locales (serie y COM virtuales sobre USB), en el menú aparece un nuevo puerto descrito por un nombre complejo que contiene una dirección de red. Por ejemplo *//192.168.10/Arduino*. Esto, obviamente si el sistema RandA esta encendido y está conectado en red, a través del puerto ethernet, disponible en Raspberry, o a través de un dongle WiFi, también conectado a Raspberry Pi.

En realidad podrían tener distintos puertos en la lista, si hubiese más sistemas RandA conectados en red, por lo que podremos también cargar sketch sobre distintos sistemas. Además, si no tuviéramos asignado establemente sobre el router, una dirección de red local para RandA, podremos utilizar este IDE modificado para controlar la dirección asignada por el router; para después conectarnos con un client SSH como MobaXterm o vía browser al server WEB instalado sobre RandA.

Para hacer una primera prueba, podemos utilizar un sketch nuestro escrito para una Arduino estándar, o uno de los ejemplos listados en el IDE. Una vez seleccionado el puerto remoto y subido al Arduino, el sketch estará operativo como siempre. Por ejemplo, si hubiéramos utilizado el sketch de blinking (led que parpadea), podremos ver el LED sobre la tarjeta RandA parpadear. Recordemos, de hecho, que la tarjeta RandA es completamente compatible con Arduino Uno.

¿Pero y si quisiéramos cargar un sketch que utiliza el puerto serie? No tengáis miedo, activando la consola del IDE, como siempre, veremos los datos llegar (o salir) como si Arduino estuviese conectado localmente. De hecho, en RandA el puerto serie ha sido completamente puesto en remoto; por tanto podemos interactuar con Arduino completamente.

Hasta aquí hemos comprobado que, sin tener que utilizar shield apropiados, Arduino se encuentra conectado en red porque se apoya en las características de networking de Raspberry Pi. Lo contrario, es posible subir el sketch, de otra manera con el uso de los shield de red. Aunque esto es un buen resultado, es solo una pequeña parte del potencial del sistema RandA.

Sobre la tarjeta RandA podemos conectar uno de los numerosos shield disponibles para Arduino, para gestionar o ampliar las conexiones de E/E: extensores de E/S, sistemas de relé, sistemas opto-aislados, gestión de motores, etc.

Pero se puede hacer mucho más y

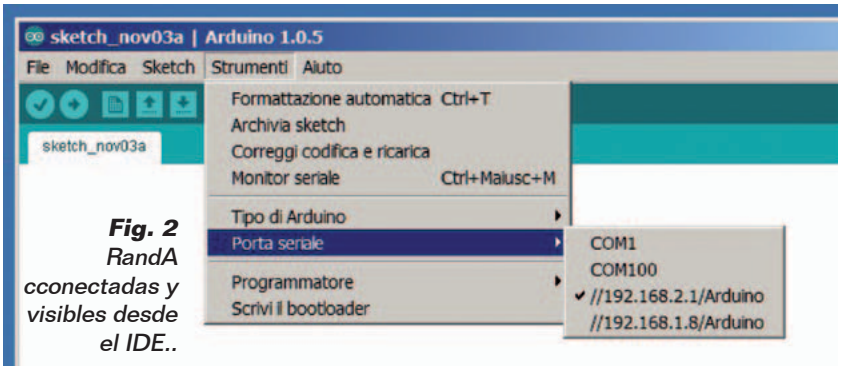


***Fig. 2** RandA cconectadas y visibles desde el IDE..*

superar, en parte, las limitaciones hardware de Arduino. De hecho se puede pensar utilizar Raspberry Pi para hacerle ejecutar las tareas. Para hacer esto es necesario que Arduino pueda ejecutar comandos sobre Raspberry Pi y leer la respuesta de estos.

Para tal objetivo se ha preparado una biblioteca Arduino incluida en el IDE modificado. Esta biblioteca (RAComm) prevé la posibilidad de lanzar comandos sobre Raspberry y recibir la respuesta. Pero permite también abrir un archivo para escribir o leer.

Ya que, mientras dialoga con Raspberry Pi, Arduino utiliza el puerto serie, este no está disponible para el dialogo con el IDE. Por lo que no es posible utilizar la consola del IDE. Por este motivo se ha previsto la posibilidad de abrir una consola sobre Raspberry Pi para escribir mensajes, para leer entradas o simplemente para el debugging. Pero esta consola, realizada en modo X terminal, necesita de una conexión con Raspberry (en red con protocolo SSH o dirigido por terminal) para ser visible.

Veamos ahora un ejemplo práctico de esta sinergia entre Arduino y Raspberry Pi: usamos un comando Linux útil que permite hacer una petición (GET) de un recurso sobre red, el comando "curl".

Probamos a utilizarlo para pedir a la web "openweathermap.org" datos atmosféricos relativos a una localidad. Siguiendo las indicaciones de la web, sabemos que la petición se realiza en el formato que se puede ver en **Fig. 3**, donde aparece también su respuesta. En el ejemplo, hemos utilizado **Roma**, pero la petición puede ser realizada para cualquier localización mundial.

En modo base, los datos son proporcionados en formato JSON; pero se pueden tener también en formato XML. El sketch, que se

curl -i http://api.openwearhermap.org/data/2.5/weather?q=Roma,it

```
HTTP/1.1 200 Ok
Server: nginx
Date: Sun, 02 Feb 2015 14:56:22 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Source: redsis
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: GET, POST

{"coord0":{"lon":12.48,"lat":41.89},"sys":{"type":1,"id":5851
,"message":0.1529,"country":"IT","sunrise";1414907053,"sunset
":1414944182},"weather":[{"id":800,"main":{"temp":292.84,"pre
ssure":1021,"humidity":63,"temp_min":292.15,"temp_max":293.15
},"wind":{"speed":3.6,"deg":280},"clouds":{"all":0},"dt":1414
938000,"id":3169070,"name":"Roma","cod":200}
```

Comando Linux y respuesta

Nota: La respuesta está truncada en un registro de 64 caracteres máximo. por el código del sketch, pero en realidad el dato en formato JSON sin retorno de carro.

***Fig. 3** - Comando curl en ejecución.*

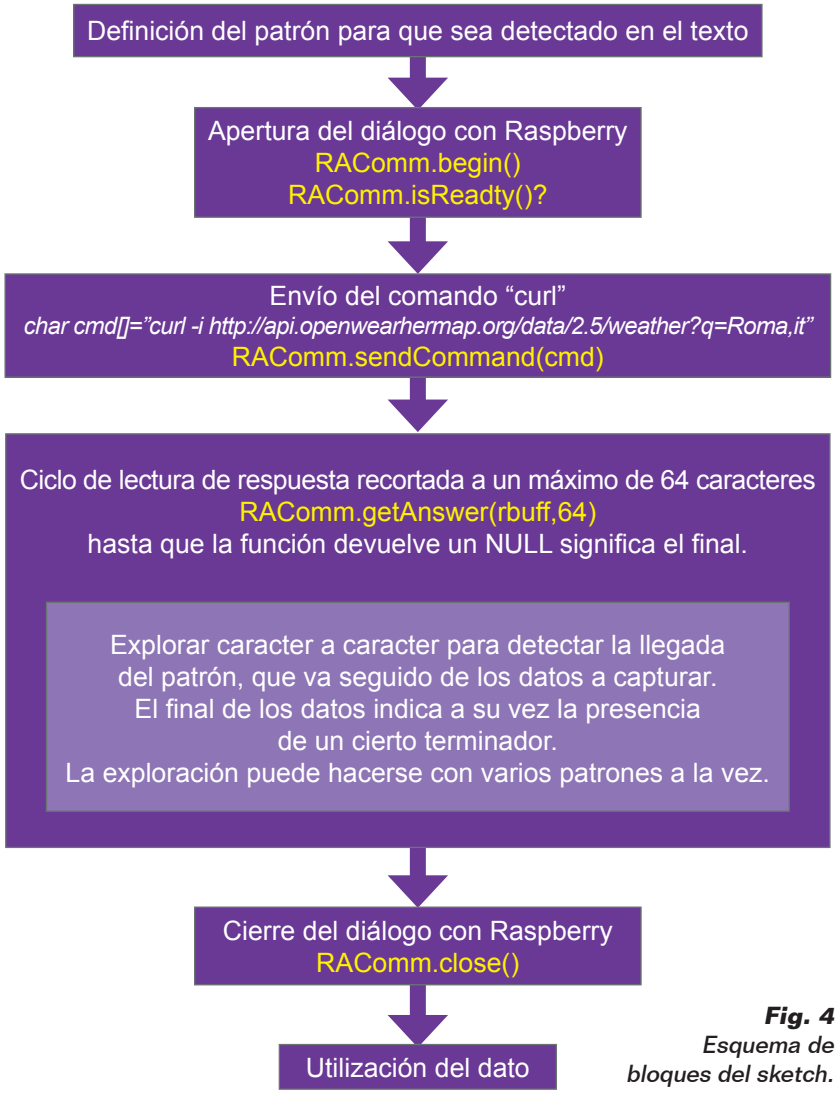


***Fig. 4** Esquema de bloques del sketch.*

Definiciones generales

```
#include "RAComm.h"       // Use Raspberry command library

#define CITY "Roma,it"

// Raspberry command to send a request to a URL
#define CMD1 "curl -i http://api.openweathermap.org/data/2.5/weather?q=" CITY

#define DEB true         // debug flag
#define LENBUFF 64       // buffer length for answer reading

RAComm Cmd;              // Cmd is the RAComm class

char rbuff[LENBUFF];     // buffer for Raspberry command answer

int flagStatus=0;        // signaling status using standard Arduino led
int led;
```

Funciones setup y loop

```
void setup()
{
  initLed();                            // standard Arduino led used as flag

  createPattArray(5);                   // define an array of patterns
  setPattern(0,"\"temp\":",',',10);     // define each pattern
  setPattern(1,"\"temp_min\":",',',10);
  setPattern(2,"\"temp_max\":",'}',10);
  setPattern(3,"\"humidity\":",',',10);
  setPattern(4,"\"pressure\":",',',10);

  getWeather();

}

void loop() // just signaling status (blinking: n times with millisec of period)
{
  switch (flagStatus) // 1: no dialog with Raspberry ; 2: error in dialog
  {
    case 1 : blinking(50,100);break;
    case 2 : blinking(10,500);break;
  }
  delay(2000);
}
```

***Fig. 5a** - Cuerpo del sketch.*

Función principal

```
int getWeather()
{
  boolean ok;
  Cmd.begin();                      // Raspberry command dialog initialization
  delay(500);                       // delay (just for tidiness)
  for (int i=0;i<20;i++)            // Max 20 checks. If not ready something is wrong!
     {ok=Cmd.isReady();if (ok) break;Cmd.begin();delay(100);}
  if (!ok) {noListen();return -1;}            // if no dialog no action

  Cmd.openFileWrite("/home/pi/wheather.log");  // open file to record data
  ok=Cmd.sendCommand(CMD1);                    // send command to get wheter
  if (!ok) {error();return -1;}
  while (Cmd.getAnswer(rbuff,LENBUFF)!=NULL)   // read ans. 64 bytes at time
  {
    if (DEB){Cmd.writeRec(rbuff);}             // if DEB write all answer on the file
    checkPattern(pattarray.ptn,rbuff);         // check every patterns and collect data
  }
  // Write data on file: temperature,tem. min.,temp. max.,humidity,pressure
  char rec[30];                          // use buffer to create record for file
  float v=0;
  char sval[10];                         // read temperature as float and get new string
                                         // because Atmel compiler doesn't use %f format
  v=getPattValN(0);v=v-273.15;dtostrf(v,5,1,sval);
  snprintf(rec,30,"Temperature: %s C",sval);Cmd.writeRec(rec);
  v=getPattValN(1);v=v-273.15;dtostrf(v,5,1,sval);
  snprintf(rec,30,"Temp-min   : %s C",sval);Cmd.writeRec(rec);
  v=getPattValN(2);v=v-273.15;dtostrf(v,5,1,sval);
  snprintf(rec,30,"Temp-max   : %s C",sval);Cmd.writeRec(rec);
  snprintf(rec,30,"Humidity   : %s%%",getPattVal(3));Cmd.writeRec(rec);
  snprintf(rec,30,"Pressure   : %s P",getPattVal(4));Cmd.writeRec(rec);
  Cmd.close();                                  // close dialog
}
```

proporciona como ejemplo en la biblioteca, se llama "ReadWeb" y estaría formado por pocas líneas de código si no fuera por la parte que controla la extracción de los datos desde el confuso texto restituido por la web. De hecho, además del código del protocolo HTTP, también los datos proporcionados son numerosos. En este punto debemos tener presente la limitación de memoria de Arduino. De hecho no se puede pensar en memorizar toda la respuesta en un buffer; además el buffer serie es de 64 byte y no es oportuno leer datos mayores para evitar la pérdida de los mismos. Por lo que debemos pedir la respuesta en pequeñas partes y recibir carácter por carácter para identificar el inicio del dato que queremos extraer. En **Fig. 4** se representa un esquema en bloques del sketch. En **Fig. 5a** y **Fig. 5b** se muestran las partes esenciales del sketch. Como se puede ver, la petición está hecha "un tantum", y para repetirla es necesario resetear Arduino; de hecho este sketch es demostrativo, mientras para un uso real, probablemente, sería necesario repetir la petición a intervalos regulares. Para hacer esto basta añadir otro comando que regule la alarma y después un comando de "shutdown". O por ejemplo para provocar la lectura de los datos meteorológicos cada hora basta insertar:

- *RAComm.sendCommand("SetRestartAt -sd 00")*
- *RAComm.sendCommand("sudo shutdown -h now")*

El primero regula la alarma al pasar la hora, mientras el segundo provoca el apagado del sistema. Recordamos que en RandA el "shutdown -h" provoca la interrupción de la alimentación al final del proceso de cierre. Además el sketch escribe sobre un archivo toda la respuesta, realizando una especie de log. Para hacer esto se utilizan funciones de biblioteca que permiten abrir un archivo (¡uno solo a la vez!) para escribir o para leer.
Los comandos utilizados en el ejemplo son:

- *RAComm.openFileWrite("home/pi/weather.log")*
- *RAComm.writeRec(rbuff)*

El archivo se cierra automáticamente por el comando de cierre del dialogo con Raspberry (aunque, estrictamente hablando, existe un comando especializado).
Los datos extraídos de la respuesta se pueden ver en la **Fig. 6**. Para la temperatura se ha realizado una conversión de temperatura absoluta a temperatura en grados centígrados (T absoluta -273,15).
Los datos se guardan en el archivo pero para realizar una aplicación práctica se podrían usar por ejemplo para activar o regular un sistema de calentamiento o enfriamiento a través de las salidas digitales o analógicas de Arduino.
Como se ha dicho, la parte más consistente del sketch es aquella relativa a una problemática que no conlleva la interacción Arduino-Raspberry, sino más bien la superación de las limitaciones de la RAM de Arduino. La respuesta se separa en record de máximo 64 caracteres, pero el pattern a medir (por ejemplo '"temp":') puede terminar separado en dos líneas.
Aunque si el problema no se refiere directamente a este artículo, daremos aquí una breve descripción del algoritmo adoptado.
La problemática ha sido generalizada y por tanto es posible utilizarla en otras ocasiones para leer, por ejemplo, un texto HTML en busca de pattern particulares en una lógica: clave-valor.
Antes de nada se definen una estructura "pattern" y una matriz de estos. Cada elemento de la matriz contiene el pattern a buscar que funciona como inicio del dato a leer, y un carácter que indica el fin de este. Además están definidas algunas variables usadas por el algoritmo. El algoritmo funciona confrontando en paralelo todos los pattern definidos, enunciando los caracteres individuales. Cuando encuentra un pattern, empieza a leer el dato de manera independiente de los otros pattern. Los datos son memorizados (como cadenas) en un buffer de la estructura correspondiente.

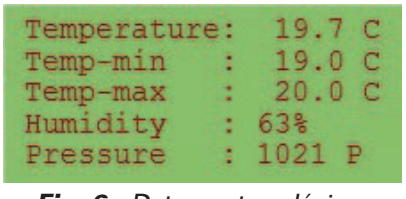


***Fig. 6** - Datos meteorológicos extraídos.*

Antes de pasar a otro ejemplo de uso de la biblioteca RAComm, es el caso de aclarar que en *RAComm.sendCommand()* es posible insertar todos los comandos estándar de las distribuciones Linux (Raspbian), los programas que hemos predispuesto para RandA y cualquier script o ejecutable que vosotros implementéis.
Hemos visto ya el uso de un comando predispuesto para RandA: *SetRestartAt* (es un ejecutable C); ahora es el momneto de listar brevemente todos los demás comandos, que podréis ver en la **Tabla 1**, pero que es también posible visualizar también por consola Raspberry Pi tecleando el comando "commands". Añadiendo *-h* se visualizan también todos los help. En realidad algunos de estos, como ArduLoad, ArduIO y ArduInterrupt, están pensados para un uso Raspberry-centrico, en cuanto proporcionan un acceso a las funcionalidades Arduino por parte de script Linux.
El segundo ejemplo que veremos se refiere al envió de un e-mail: supongamos que queremos detectar la presencia de una señal importante y avisar en consecuencia a alguien del evento. La parte relativa a la detección de la condición es típicamente un deber para Arduino. Por ejemplo, podremos detectar la superación de un umbral analógico o una condición ON/OFF; pero enviar un e-mail con Arruino es una tarea difícil,

Tabla 1

Comando	Significado
Arduload	Carga un sketch compilado (.hex) guardado sobre Raspberry
ArduInterrupt	Se bloquea hasta una condición
Sobre los pines de Arduino	Si blocca fino ad una condizione su pin di Arduino
ResetRandA	Reset de Arduino
GetRTC	Lee el reloj de RandA
SetRTC	Ajusta el reloj de RandA
SetRestartAt	Define la alarma para el encendido
ResetAlarm	Elimina la alarma eventualmente activada
SetSysClock	Ajusta el reloj del sistema utilizando el RTC
SendMail	Envía un e-mail

sobre todo debido a los server SMTP que utilizan los protocolos protegidos, cuya implementación con Arduino es imposible.
El comando a utilizar podría ser, por ejemplo:

```
RAComm.sendCommand("
SendMail mailto=\"pippo@pluto.it\"
subject=\"Rilevazione\"
filemess=\"/home/pi/messaggio.txt\" ")
```

El comando ha sido partido en más líneas, exclusivamente por comodidad de impresión. El pattern \" esta para indicar la presencia de dobles comillas en el interior de una cadena (el comando a pasar como argumento a la función). El texto del mensaje es automáticamente formado por "SendMail" tomándolo del archivo en el cual se ha apoyado.
El comando prevé la posibilidad de enviar también simplemente un texto corto, en vez del texto contenido en un archivo, de este modo:

```
RAComm.sendCommand("
SendMail mailto=\"pippo@pluto.it\"
subject=\"Rilevazione\"
message=\"Valore : 320\" ")
```

También es posible añadir un adjunto, o tener más destinatarios.

Definiciones

```
#include "RAComm.h"     // Use Raspberry command library

#define analog 4        // analogic pin used
#define LOGFILE "/home/pi/values.dat" // log file name
#define TEXTMAIL "/home/pi/mailval.txt" // log file name


#define Q "\""                          // escape character for quote(")
#define SENDTO "revista@nuevaelectronica.com"              // addressee (ex.: smith@gmai.com)
#define SUBJECT "Value"                 // subject

//Commands:

// Raspberry command to get timestamp
#define CMD1 "GetRTC -s"

// Command that uses "tail" linux command to extract last 5 lines from log whereby making text to send
#define CMD2 "tail -n 5 " LOGFILE " > " TEXTMAIL

// Raspberry command "SendMail" text (NB. text in quote has to be inserted using escape char)
#define CMD3 "SendMail mailto=" SENDTO " subject=" Q SUBJECT Q " filemess=" Q TEXTMAIL Q

// Raspberry command for alarm setting (use in sprintf statment)
#define CMD4 "SetRestartAt -sd %d %d"

// Shutdown command
#define CMD5 "sudo shutdown -h now"

RAComm Cmd;          // Cmd is the RAComm class
```

***Fig. 7** - Sketch SendMail (definiciones principales).*

Pero para mayores detalles se remite al help del comando.
En este punto es el caso, sin embargo, de precisar que es necesario inicializar el archivo "/home/pi/bin/Mail.properties" con los datos propios; es decir, con el nombre de usuario y el password reconocidos por el servidor SMTP utilizado y la dirección del server mismo. Además hay que tener presente que quien no está habituado a utilizar un cliente de correo electrónico (como Outlook o Thunderbird) podría encontrar el servidor de Gmail no habilitado para recibir comandos de envió; en tal caso es necesario hacer los ajustes del tipo en la web de Gmail.
En el ejemplo, incluido en la bibioteca, llamado "SendMail", se detecta periódicamente un valor analógico, el cual junto al *timestamp* detectado por el RTC se memoriza sobre un archivo y después es enviado a una dirección e-mail. Fijaros que no es enviado solo el último dato leído: se envían por e-mail los últimos 5 registros. Para hacer esto es utilizado el comando Linux "tail" que extrae los últimos "n" registros desde un archivo.
En la **Fig. 7** se visualizan solo las principales definiciones presentes en el sketch.
Con los ejemplos mostrados, el potencial de la biblioteca de conexión con Raspberry Pi debería resultar evidente. Además tener presente que, en caso de error lógico (por ejemplo sketch que provoca inmediatamente el shutdown al arrancar el sistema), abriendo el puente JP1 es posible deshabilitar el dialogo con Raspberry Pi; JP1 tiene el objetivo de informar a Raspberry Pi la intención de comunicar.
El script que escucha la señal se lanza al arrancar (en el archivo */etc/rc.local*) y se llama StartListenCmd; se puede lanzar también con un archivo de log. Más simple, en caso de graves problemas de debug se puede lanzar directamente el programa C que dialoga y que se llama *ExecSCmd* (después de haber deshabilitado *StartListenCmd*). *ExecSCmd* puede ser lanzado con el flag de debug *-s*, con el cual hace eco sobre la consola de todo lo que pasa en serie. La fuente de *Execscmd* se encuentra, junto a las otras fuentes de los programas para RandA, en:

```
"/home/pi/workspace/cworkspace"
```

Hay además otra posibilidad para aprovechar en el enfoque Arduino-centrico: se puede tener Arduino siempre alimentado, actuando sobre el puente SW2 que lo separa de la gestión del interruptor electrónico.
Arduino, privado de su fuente de alimentación lineal y del USB, consume poquísimo y se puede poner, incluso solo, en condición de bajísimo consumo. En esta condición, Arduino puede encender Raspberry Pi dando un impulso a su pin D4 (siempre que el correspondiente puente JP2 esté cerrado). Otro ejemplo contenido en la biblioteca hace ver este uso, y enciende Raspberry Pi si sobre su entrada analógica A0 está presente un valor superior a 300.

ENFOQUE RASPBERRY-CÉNTRICO

Quien está familiarizado con el sistema operativo Linux, encon-

trará seguramente más simple utilizar Raspberry como base de programación. Para este objetivo se puede utilizar un entorno de desarrollo para C potente como "codeblocks" que hemos instalado sobre Raspberry Pi, o el lenguaje Python por el cual hemos instalado el entorno de desarrollo "idle". Ya que sobre Raspberry Pi está instalado también Java, es posible utilizar también este potentísimo lenguaje. De hecho están presentes ya programas Java utilizados en RandA, como el comando de SendMail o los servlet presentes en el servidor Web del cual hablaremos más adelante. No tenemos sin embargo instalado un entorno de desarrollo para Java ya que no hemos cumplido aún la disponibilidad de IDE gratuitos para un hardware reducido como Raspberry PI. Aunque puede instalar desde la web paquetes Debian, Eclipse no está en una versión suficiente debido a problemas tanto de instalación como de rendimiento.
Los programas Java completos de fuente son catalogados en:

```
"/home/pi/workspace/jworkspace"
```

Los programas (o los script) realizados pueden ser también utilizados en el enfoque Arduinocéntrico, como hemos visto ya.
O pueden utilizar el puerto serie para comunicar con sketch predefinidos para gestionar la comunicación (es decir, fuera de la biblioteca RAComm). De este modo Arduino se convierte en un potente periférico programable de Raspberry Pi.
El comando "Arduload", que hay que añadir en nombre del archivo ejecutable (.hex) con el path completo, permite modificar de vez en cuando el comportamiento de Arduino, porque "instala" sobre Arduino el sketch que utiliza la aplicación. Este sketch, en formato de archivo ejecutable, puede haber sido creado con el IDE remoto o con el IDE instalado localmente sobre Raspberry Pi; de hecho ambas versiones del IDE, además de hacer la carga del sketch, guardan el ejecutable sobre Raspberry Pi en dos directorios distintos:
- */home/ArduinoUpload* para los sketch realizados localmente;
- */home/RArduinoUploads* para los sketch realizados en remoto.

Esta es otra función adicional del IDE modificado por nosotros.
Como ejemplo de aplicación que trata Arduino como periférico, basta hacer referencia al comando "ArduIO" ya listado en la **Tabla 1**. Este programa C lo encontráis también en el "cworkspace", junto a su código fuente.
ArduIO quiere ser el equivalente de la utilidad GPIO (wiringPi) de Raspberry Pi, de hecho permite gestionar los pines de Arduino desde línea de comando. Por ejemplo, para encender el LED estándar de Arduino (pin 13) es necesario:
1. definir como output el pin 13 con *ArduIO -set 13 out*;
2. poner a 1 el pin 13 con *ArduIO -wrd 13 1*.

Para apagarlo se usa el comando *ArduIO -wrd 13 0*.
Es posible utilizar las típicas modalidades previstas por Arduino: leer pines analógicos y digitales, activar/desactivar salidas digitales y regular los pines PWM. Tecleando "ArduIO -h" se obtiene la ayuda detallada. Más en detalle, en **Tabla 2** están listadas las distintas opciones.
El programa utiliza un sketch (*SerialRasp*) con el cual colabora enviándole los comandos correspondientes: es este último que activa los pines. Por tanto el sketch debe haber sido descargado sobre Arduino anteriormente. En realidad ArduIO supera este problema, cargándolo el mismo, en el caso que no estuviese presente en Arduino.
Para permitir esto es necesario:
a) que pueda identificar el sketch presente sobre Arduino;
b) que pueda encontrar el sketch en una posición prestablecida en el caso que no fuese aquel establecido y fuese necesario cargarlo.

El primer punto se resuelve fácilmente preparando el sketch en modo que, si se pregunta, responda con el propio nombre; para el segundo punto se ha preparado un directorio llamado "/home/pi/bin/sketch4cmd" de donde recogerlo. En la **Fig. 8** se evidencia la colaboración entre los dos entornos, con particular referencia al reconocimiento del sketch y a una eventual carga automática del mismo.
Pero no es necesario utilizar lenguajes compilados, como C, para realizar aplicaciones sobre Raspberry Pi que usen Arduino para entradas/salidas: se pueden utilizar también script bash (Linux) o script Python. Basta utilizar los comandos prestablecidos para RandA (**Tabla 1)** para tener la tarea muy simplificada.
En el ejemplo siguiente se muestra una pequeñísima aplicación demostrativa que lee el valor de

Tabla 2

Opción	Significado
--set n x	Set del pin n como x=inp,out,ipl (ipl:input-pullup)
-rda n	Lee pin analógico n
-rdd n	Lee pin digital n
-wra n v	Set de los pines PWM
-wrd n v	Set de los pines digitales (0/1) (Atención al pin 4: quitar el puente para utilizarlo)(El pin 4 está conectado al switch ON/OFF)
-pou n v	Pulsación sobre el pin n de frecuencia v (tone)
-poi n v	Lee duración impulso sobre pin n (v=0/1 define impulso alto o bajo)

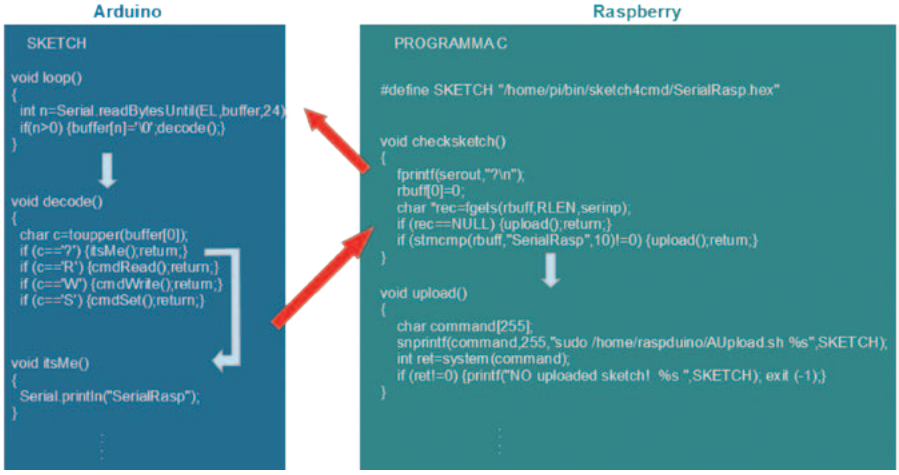


***Fig. 8** - SerialRasp colabora con ArduIO.*

tensión presente sobre el pin A1 (por ejemplo proporcionado por una fotocélula) y activa el LED estándar de Arduino en el momento en el cual el valor leído es inferior a un umbral. En la **Fig. 9a** se muestra la versión realizada con un script bash, mientras en la **Fig. 9b** es visible el equivalente en Python.

Naturalmente el script es sobretodo didáctico, pero con pequeños añadidos se le puede insertar un ciclo periódico o se le puede insertar un control horario o el envió de un e-mail de aviso.

Estos dos archivos (*ScriptExample4IO.sh* e *PyExample4IO.py*) están presentes en el directorio */home/pi/bin/examples*. Hay además otro

```
#!/bin/bash
ArduIO - set 13 out              #set LED
out="${ArduIO -rda 1}"           #value of A1 in out variable
if [[ $out == NO* ]]             #if can't read exit
  then echo "Err!"; exit 0; fi
out=${out//[^0-9]/}              #delete any character not digit
echo $out                        #so, in out just a number
if [ "$out" -lt "200" ]          #if < of trheshold
then
  echo "Switch on!"
  ArduIO -wrd 13 1               #LED on
else
  ArduIO -wrd 13 0               #LED off
  echo "Nothing to do!"
fi
```

***Fig. 9a** - Script bash para el encendido condicionado..*

```
#!/usr/bin/pyton
import os
import commands
ret=os.system("ArduIO -rds 1")
out=commands.getoutput("ArduIO -rda 1")
if out[0:1] == "NO": quit()
out=int(out)
print out
if out < 200:
     print "Switch on!"
     ret=os.system("Arduino -wrd 13 1")
else:
     print "Nothing to do"
     ret=os.system("Arduino -wrd 13 0")
```

***Fig. 9b** - Script Python equivalente al de **Fig. 9a**.*

comando que permite a Raspberry Pi utilizar el potencial de Arduino, y que puede ser utilizado en el interior de los script: ArduInterrupt. Este programa utiliza el sketch "SerialStop", presente en "/home/pi/bin/sketch4cmd", y en sustancia se bloquea hasta que una cierta condición no se cumple; la condición puede ser sobre un pin digital, cuando se convierte en 1 o 0, o sobre un pin analógico cuando el valor medido es mayor o menor de un cierto umbral.

Por ejemplo, "ArduInterrupt -ana 2 lt 200", se bloquea hasta que sobre el pin analógico 2 de Arduino el valor no baja de 200. Es evidente que el uso más oportuno de este comando es en el interior de un script donde las instrucciones siguientes realizan una acción. Además el script será probablemente lanzado en background.

¿Pero cómo desarrollar aplicaciones sobre Raspberry de manera eficiente? Con el enfoque Arduino-céntrico hemos usado una ubicación remota (cualquier PC conectado en red local) para desarrollar sketch que pudiesen utilizar también el potencial de Raspberry Pi; ahora, invirtiendo el punto de vista es aún posible utilizar una ubicación remota (siempre nuestro PC conectado en red local), esta vez utilizando una consola remota Linux sobre protocolo SSH pero con la posibilidad añadida de un ambiente gráfico. Para esto se puede utilizar un software que permita actuar también como terminal X Window (que es el ambiente grafico base de Linux). Uno de estos es MobaXterm (gratuito y descargable desde la web *http://mobaxterm.mobatek.net/*).

En la **Fig. 10** se muestra una situación típica en la que junto a la consola MobaXterm, se superpone una ventana sobre el system file y una ventana codeblocks IDE para la programación C.

Web Application y CGI

La web nace como proveedor de páginas ricamente estructuradas y multimedia, pero estáticas. No por casualidad nace en ambiente académico, donde ha sido pensado como un inmenso libro distribuido. Enseguida, sin embargo, se ha comenzado a buscar interactuar con las páginas insertando datos y esperando respuestas. Para hacer esto es necesario que el servidor web tenga la posibilidad de activar programas que recibiendo la entrada sepan construir la respuesta, en formato HTML, que el servidro pueda responder al solicitante. Los programas que pueden hacer esto son integrados en aquel que es llamado CGI (Common Gateway Interface). Siguiendo esta interfaz el servidor WEB debe saber interceptar la petición como petición de una elaboración distinta de la petición de una página. En el primer caso debe saber que programa activar y pasarle la petición; los programas que son utilizados para esta funcionalidad son, en general, intérpretes de script.
El aumento de la petición de interacción, como por ejemplo el uso de la tecnología AJAX que interactúa en background, ha llevado a integrar la capacidad de elaboración más internamente al servidor Web; de aquí el paso a los Web Applications Server. Las dos tecnologías distintas: dot net y Java, rivalizan en este campo; la primera en exclusivo ámbito Microsoft, mientras la otra es más universal.
En el caso de Java, se habla de una integración formada por agiles thread llamados Servlet o de un uso suyo más complejo, parecido a los script PHP, representado por las paginas JSP. Con Servlet y JSP, la interacción es mucho más fluida porque no hay procesos enteros que activar, ni script que interpretar (de hecho las paginas JSP son automáticamente compiladas en Servlet).
En el caso de interacción blanda, la complejidad de la programación de los Servlet puede ser excesiva respecto a la tarea, razón por la cual a veces es conveniente el uso de la tecnología CGI.
Con la tecnología CGI, el programa (script) es activado llamando el recurso con el nombre del script. Esto va colocado con una referencia precisa en la estructura de la web. El script revive los parámetros de la petición a través de la variable de ambiente QUERYSTRING y su salida será restituida desde el servidor Web por el solicitante.

Quien está más habituado a utilizar Linux, o en todo caso los modernos sistemas operativos, preferirá este enfoque Raspberry-céntrico, porque permite una mayor flexibilidad y el eficiente uso del potencial del sistema RandA. Sin embargo quien está más habituado al ambiente de Arduino, preferirá el anterior enfoque.
En realidad es útil también mezclar los dos puntos de vista según la aplicación. Pero, sobre todo, entendemos RandA un sistema ideal para el aprendizaje y el primer enfoque a la informática y a los sistemas digitales, en cuanto es posible cubrir, con ello, una increíble cantidad de argumentos y de ambientes operativos.

EL SERVIDOR WEB

El software suministrado no acaba con lo que apenas hemos descrito; de hecho hemos pensado añadir también un Servidor Web, que permite una gestión básica del sistema también en red geográfica. De hecho hasta ahora hemos definido el uso del enfoque remoto exclusivamente a la red local, tanto para la tipología del software de red utilizado, como por motivos de seguridad. Con un Servidor Web, es sin embargo posible superar este límite, para definir un puerto de salida sobre el router local ("port forwarding"), o tener una dirección de red visible al exterior.
El servidor software utilizado no es solo un servidor Web sino un "Web Application Server". Este modo utiliza un enfoque más eficazmente orientado a la interacción. El servidor se basa sobre uno de los más conocidos software gratuitos: Tomcat. Desafortunadamente un software de este tipo prevé el uso de Servlet Java (o script JSP) para las aplicaciones interactivas. Pero la complejidad del software utilizado se compensa por la notable eficiencia de las aplicaciones: basta pensar que todos los servidores profesionales utilizan este tipo de entorno o su concurrente "dot-net" (ver recuadro en esta página).
Pero si no sois expertos de Servlet o JSP no tengáis miedo porque el servido ha sido preparado para gestionar también los script CGI, es decir, script Linux o Python; en este modo podéis construiros vuestras aplicaciones interactivas y utilizarlas desde un browser en cualquier sitio del mundo.
El Tomcat instalado está en la versión 7 en el directorio: "/home/apache-tomcat-7.0.47"
En este mismo directorio están presentes dos script: *startWebS.sh* e *stopWebS.sh* para arrancar y cerrar el servidor web. El primero es llamado desde el interior del archivo de sistema de inicio "/etc/rc.local". Si se quiere deshabilitar el arranque automático del servidor Web, basta comentar este comando en el susodicho archivo de sistema.
El servidor Tomcat ha sido puesto en escucha sobre el puerto 80 (en vez del 8080) actualizando el archivo: "/home/apache-tomcat-7.0.47/conf/server.xml".
En el interior del directorio de Tomcat está incluido todo el servidor Web: desde las aplicaciones a las paginas estáticas. En particular, el directorio "…/webapps/ROOT" contiene la página de arranque "index.html". Sin embargo la aplicación para la gestión de Raspberry Pi y Arduino está toda contenida en el directorio "…/webapps/RandA".
Para quien quiera cargar aplicaciones propias o aplicaciones de terceras partes, la forma más simple es utilizar los archivos del archivo .war (con los que normalmente se distribuyen las aplicaciones web Java) e instalarlos a través de la consola del manager Tomcat.
A la consola se accede desde la dirección "http://…../manager"

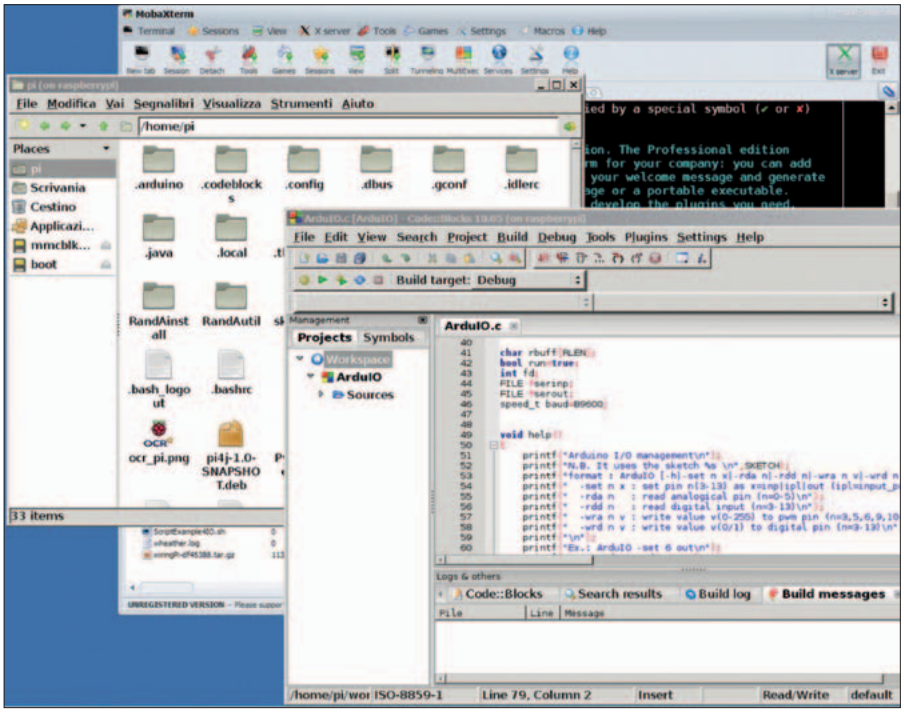


***Fig. 10** - Entorno de gestión Raspberry-céntrico.*

y está protegida por nombre de usuario (tomcat es el predeterminado) y password (tomcat es el predeterminado). El archivo .war puede estar también en el ordenador conectado al servidor web; de hecho la consola permite hacer automáticamente la descarga y la instalación en un solo paso. Por tanto en conjunto la instalación de aplicaciones se hace de forma remota.
Quien quisiera sin embargo utilizar paginas HTML simples puede colocarlas en el directorio ROOT, donde estarán inmediatamente disponibles. Obviamente podrán ser creadas también bajo directorios de ROOT.
Sin embargo en caso de uso de la modalidad CGI, los script bash se colocan en el directorio:

```
"/home/apache-tomcat-7.0.47/webapps/
ROOT/WEB-INF/cgi"
```

Pero, atención, son referenciados como: http://...../bin-cgi/nombre-delscript.
Ejemplo: *http://192.168.1.8/bin-cgi/testcgi2.sh*. Los archivos "testcgi.sh" y "testcgi2.sh" presentes en el directorio …../ROOT/WEB-INF/cgi, son dos archivos de ejemplo que muestran el uso de script bash sencillos para el dialogo web y el lanzamiento de comandos Linux.
En el caso en que se quiera usar un intérprete de script distinto, es necesario editar el archivo "/home/apache-tomcat-7.0.47/conf/web.xml" (bloque <servlet> parametro "executable").
La página de inicio (para visualizarla basta insertar en el browser la dirección de RandA), da acceso a la aplicación de gestión y a una consola web (con suerte) para el acceso a Linux incluso en red geográfica (aplicación de terceras partes).
La aplicación de gestión no está por el momento protegida por password, pero se puede proveer fácilmente editando el archivo "/home/apache-tomcat-7.0.47/webapps/RandA/WEB-INF/web.xml", y quitando el comentario del bloque de autenticación.
Mientras la consola web tiene la dupla usuario, password: randa, randa. La aplicación de gestión de acceso a las siguientes paginas/ aplicaciones:
- carga sketch sobre Arduino;
- consola de Arduino;
- gestión I/O Arduino;
- definición de la alarma;
- ajuste del reloj;
- apagado de RandA.

Las aplicaciones arriba listadas permiten tener un acceso remoto al sistema RandA incluso desde red geográfica, pero el servidor ha sido instalado pensando sobretodo como base para la personalización por parte de los usuarios, quizás recurriendo a script CGI.

CONCLUSIONES

Hemos buscado proporcionar un sistema preparado y funcional, disponible para ser utilizado en aplicaciones de distintos tipos, también con el añadido de hardware especializado; todo manteniendo una visión didáctica, para aficionados y desarrolladores, comentando el máximo posible el software producido para facilitar personalizaciones y mejoras.

(191057)

Este proyecto puede ser fácilmente realizado por cualquiera que tenga un mínimo de experiencia en el montaje manual de componentes SMD y disponga del equipo necesario. La placa RandA esta también disponible ya montada y probada (incluidas las piezas pequeñas) al precio de 39,00 Euros. Está disponible también la placa Raspberry PI Tip B+ (cod. 8111284RS) al precio de 36,00 Euros.

Precios IVA incluido sin gastos de envío.
Puede hacer su pedido en:
www.nuevaelectronica.com
pedidos@nuevaelectronica.com