Contenidos dinámicos en micro servidores web para sistemas embebidos

E. Sergio Burgos[1] and Leonardo Giovanini[23]

[1] Universidad Tecnológica Nacional, Facultad Regional Paraná,
Almafuerte 1033, 3100 Paraná, Entre Ríos, Argentina
sergioburgos@frp.utn.edu.ar
[2] Universidad Nacional del Litoral, Facultad de Ingeniería y Ciencias Hídricas
[3] Consejo Nacional de Investigaciones Científicas y Tecnológicas
lgiovanini@fich.unl.edu.ar

Resumen El uso de servidores web en sistemas embebidos suele hacer uso de páginas html generadas dinámicamente para interactuar con el usuario. En este trabajo se presenta como alternativa la generación dinámica de contenidos XML como origen de datos para aplicaciones Adobe Flash©, lográndose reducir el flujo de información entre el cliente y el servidor; mejorando la experiencia de usuario.

Keywords: micro web server, sistemas embebidos, control, internet, tcp/ip

1. Introducción

La creciente potencia que se observa en los microcontroladores existentes en el mercado y el uso cada vez más común de redes TCP/IP e Internet han ocasionado que cada vez sea más común el uso de servidores web en sistemas embebidos. En general el objetivo es permitir la interacción con el usuario a través de páginas webs almacenadas en el microcontrolador para, de esta forma, modificar parámetros del sistema, controlarlo o presentar el estado del mismo[5]. Es por esto que los servidores web embebidos permiten trabajar tanto con contenido estático (páginas web cuyo contenido no cambia) como dinámico (páginas web cuyo contenido se conforma como resultado de algún tipo de procesamiento). Debido a los pocos recursos de los que en general se dispone en sistemas embebidos la generación de contenidos dinámicos está asociada a implementaciones no estándares que permiten generar contenido html. El problema que se suele presentar al trabajar con estos entornos es que se trata de lograr la mayor interacción posible con el usuario utilizando la menor cantidad de recursos del sistema. Por esto, la mayoría de las páginas solo incorporan los elementos definidos en el estándar html, imágenes y, ocasionalmente, javascript para mejorar la experiencia de usuario.
Una alternativa para enriquecer el contenido html, es el uso de aplicaciones que se ejecutan en el navegador, tales como java applets o aplicaciones Flash©. Sin embargo, su uso, presenta un nuevo desafío, la comunicación entre la aplicación

ejecutada en el navegador web y el sistema embebido. En este trabajo se presenta la metodología utilizada para resolver este problema y los resultados obtenidos en la implementación de una aplicación web, desarrollada en Adobe Flash, para interactuar con un sistema embebido utilizando un microcontrolador LM3S2776 de la empresa Texas Instruments.
El objetivo del desarrollo era lograr una aplicación web que permitiera interactuar con los diversos periféricos incorporados en el kit y representar el estado de las entradas analógicas gráficamente. Específicamente las operaciones de interés eran controlar (encender y apagar) el led de estado, enviar texto introducido por el usuario a la pantalla LCD, borrar el contenido de la pantalla y graficar los valores presentes en las 4 entradas analógicas del conversor analógico digital más el valor de temperatura registrado en el sensor incorporado en el microcontrolador.
El tipo de control perseguido requiere el flujo de información en diferentes sentidos, pero más allá de esto la única alternativa para su solución es la ejecución de rutinas para la generación de contenido dinámico. Las acciones directas sobre el kit, tales como encender un led no requieren más que el acceso a un contenido particular que dispare la ejecución de una rutina. Para el caso del acceso a los valores correspondientes al conversor análogico/digital, estas rutinas deben retornar valores obtenidos de la conversión. La diferencia significativa se plantea para el control del LCD, ya que este involucra el uso de datos que, originados en el navegador, deben ser utilizados en el kit.
Este trabajo se organiza como se detalla a continuación. En la Sección 2, se comenta brevemente el sistema embebido utilizado para el desarrollo. En la Sección 3 se describe el servidor web utilizado para la aplicación. En la Sección 4, se describe el desarrollo realizado. Finalmente, en la Sección 5 se presentan las conclusiones.

2. El sistema embebido

El sistema embebido utilizado para el desarrollo es un kit de usos múltiples (figura 1) que incorpora una pantalla LCD de tipo carácter, un conversor digital analógico, un led indicador de estado y tiene como núcleo un microcontrolador ARM Cortex-M3 de la línea Stellaris de Texas Instruments, el LM3S2776 [3]. El microcontrolador incorpora 128 KB de memoria de programa (FLASH), 64 KB de memoria RAM, conversor analógico digital de 10 bits y puerto de comunicaciones serie (UART). Debido a que el kit utilizado carece de interfaz ethernet, para la implementación se utilizó el protocolo IP sobre línea serie (SLIP)[4] a través de la UART. Este protocolo permite transferir paquetes TCP/IP a través de enlaces de comunicación serie, para esto se utilizan códigos especiales para marcar el comienzo y fin de los paquetes junto a secuencias de escape que permiten omitir estas marcas cuando forman parte de los paquetes de datos. De este modo, desde la perspectiva de la aplicación, se tiene un comportamiento equivalente a un sistema embebido con interfaz ethernet utilizando TCP/IP.
Para llevar las condiciones de funcionamiento más cercanas al caso de interés se

Figura 1. Foto del kit de desarrollo utilizado

conectó el kit a un equipo PC que permitiera la adaptación entre el protocolo TCP/IP sobre Wi-Fi a SLIP sobre UART, lográndose de esta manera acceder al sistema desde diferentes equipos en una red local, lográndose una estructura como la mostrada en la figura 2

3. Servidor web

La implementación de stack TCP/IP utilizada como base para el desarrollo fue la librería uIP versión 1.0[2]. Si bien esta fue desarrollada originalmente para microcontroladores de 8 bit, puede ser portada a diferentes arquitecturas de modo simple. Junto a esta librería, se distribuyen un conjunto de aplicaciones a modo de demostración, entre las que se incluye un servidor web capaz de generar contenido dinámico a partir de incorporar llamadas a funciones desde el código de las páginas webs.
El servidor es capaz de procesar solicitudes de páginas html, css (hojas de estilo) y shtml, siendo esta última la extensión asociada a páginas con contenido dinámico. En lo que respecta a imágenes permite utilizar formatos jpg, gif y png. Cada uno de estos contenidos está asociado a un tipo de encabezado html, enviado por el servidor, a partir del cual el cliente (navegador) es capaz de procesar el contenido recibido.
En lo que respecta a las peticiones GET y POST del estándar HTTP utilizadas para enviar información desde el cliente, la versión original del servidor no incorpora su procesamiento, siendo posible modificar su código fuente para permitirlo.
Si bien la aplicación desarrollada depende del servidor web utilizado, debido a que la mayoría de los servidores web para sistemas embebidos disponen de, como

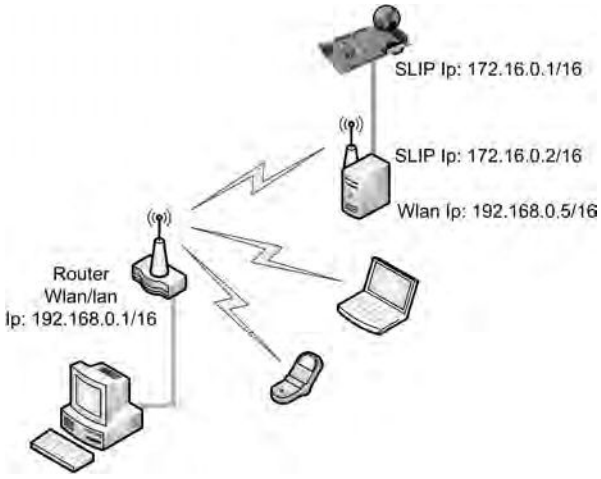

Figura 2. Esquema de la estructura de red utilizada

mínimo, las capacidades del servidor utilizado el desarrollo aquí presentado es fácilmente extensible a otros servidores web.

4. Desarrollo realizado

La clave del desarrollo consiste en utilizar lenguaje extensible de marcas (XML) para establecer la comunicación entre el servidor web y la aplicación Flash, de este modo generando dinámicamente el contenido XML y a partir de las clases incorporadas en lenguaje Action Script 3.0 es posible implementar una comunicación de modo simple.
Esto requiere modificar la implementación del servidor web y desarrollar una aplicación cliente que haga uso de estas modificaciones, estas tareas son abordadas en las secciones siguientes.

4.1. Servidor Web

El primer paso para en el desarrollo es agregar a los encabezados HTTP que incorpora el servidor un par más, uno asociado a contenidos XML (text/xml) y otro asociado a la aplicación Flash misma (application/x-shockwave-flash). De este modo los clientes al solicitar este tipo de contenidos podrán procesarlo de manera adecuada. Este tipo de modificaciones se encuentran previstas en la implementación del servidor web utilizado, para esto se incorpora un archivo (http-strings) que define el tipo de contenido a manejar y su extensión. A partir de éste se generan, utilizando un script perl (makestrings), dos archivos de código fuente (http-strings.h y http-strings.c) que incluyen el contenido de las cadenas de texto originales como vectores inicializados.
Habiendo realizado las modificaciones pertinentes es necesario modificar la lógica

del servidor (httpd.c) para que permita enviar los headers adecuados en función del tipo de solicitud, para eso es necesario modificar la función encargada del procesamiento de los encabezados tal como se detalla a continuación.

```
static
PT_THREAD(send_headers(struct httpd_state *s, const char *statushdr))
{
  char *ptr;
  PSOCK_BEGIN(&s->sout);
  PSOCK_SEND_STR(&s->sout, statushdr);
  ptr = strrchr(s->filename, ISO_period);
  if(ptr == NULL) {
    PSOCK_SEND_STR(&s->sout, http_content_type_binary);
  } else if(strncmp(http_html, ptr, 5) == 0 ||
    strncmp(http_shtml, ptr, 6) == 0) {
    PSOCK_SEND_STR(&s->sout, http_content_type_html);
  }else if(strncmp(http_xml, ptr, 4) == 0 ||
    strncmp(http_sxml, ptr, 5) == 0) {
    PSOCK_SEND_STR(&s->sout, http_content_type_xml);
  }else if(strncmp(http_css, ptr, 4) == 0) {
    PSOCK_SEND_STR(&s->sout, http_content_type_css);
  } else if(strncmp(http_png, ptr, 4) == 0) {
    PSOCK_SEND_STR(&s->sout, http_content_type_png);
  } else if(strncmp(http_gif, ptr, 4) == 0) {
    PSOCK_SEND_STR(&s->sout, http_content_type_gif);
  } else if(strncmp(http_jpg, ptr, 4) == 0) {
    PSOCK_SEND_STR(&s->sout, http_content_type_jpg);
  }else if(strncmp(http_swf, ptr, 4) == 0) {
    PSOCK_SEND_STR(&s->sout, http_content_type_swf);
  }else {
    PSOCK_SEND_STR(&s->sout, http_content_type_plain);
  }
  PSOCK_END(&s->sout);
}
```

El siguiente paso es agregar un nuevo modo de procesamiento para las páginas dinámicas de forma que se distingan las invocaciones de rutinas que generen contenido dinámico html (shtml) de las que generen contenido dinámico xml (sxml). Esta modificación también debe realizarse en el código fuente del servidor web (httpd.c) como se muestra a continuación.

```
static PT_THREAD(handle_output(struct httpd_state *s))
{
  char *ptr;
  PT_BEGIN(&s->outputpt);
  if(!httpd_fs_open(s->filename, &s->file)) {
```

```
    httpd_fs_open(http_404_html, &s->file);
    strcpy(s->filename, http_404_html);
    PT_WAIT_THREAD(&s->outputpt, send_headers(s, http_header_404));
    PT_WAIT_THREAD(&s->outputpt, send_file(s));
  } else {
    PT_WAIT_THREAD(&s->outputpt,
   send_headers(s,
   http_header_200));
    ptr = strchr(s->filename, ISO_period);
    if(ptr != NULL && strncmp(ptr, http_shtml, 6) == 0) {
      PT_INIT(&s->scriptpt);
      PT_WAIT_THREAD(&s->outputpt, handle_script(s));
    }
    else if(ptr != NULL && strncmp(ptr, http_sxml, 5) == 0) {
      PT_INIT(&s->scriptpt);
      PT_WAIT_THREAD(&s->outputpt, handle_script(s));
    }
    else {
      PT_WAIT_THREAD(&s->outputpt, send_file(s));
    }
  }
  PSOCK_CLOSE(&s->sout);
  PT_END(&s->outputpt);
}
```

Realizadas estas modificaciones el servidor es capaz de reconocer la extensión sxml y asociarla a contenido dinámico enviando los encabezados HTTP de modo correcto y ejecutando las funciones invocadas desde el archivo solicitado. Solo falta incorporar rutinas capaces de generar las salidas xml deseadas en función de la solicitud. A modo de ejemplo se muestra la implementación de una rutina que genera un archivo XML con los valores de las entradas del conversor.

```
static unsigned short generate_analog(void *arg)
{
  unsigned long sensorValue[4];
  unsigned long tempValue;
  unsigned int sendCount;
  unsigned char i;
  adc_getAnalogInput(sensorValue);
  adc_getTempSensor(&tempValue);
  sendCount = snprintf((char *)uip_appdata,
       UIP_APPDATA_SIZE,
       "<datos>\n");
  for(i=0; i < 4; i++)
    sendCount += snprintf((char *)(uip_appdata + sendCount),
     UIP_APPDATA_SIZE-sendCount,
```

```
    "<entrada><valor>%lu</valor></entrada>\n",
    sensorValue[i]);
  sendCount += snprintf((char *)(uip_appdata + sendCount),
      UIP_APPDATA_SIZE-sendCount,
      "<entrada><valor>%lu</valor></entrada></datos>\n",
      tempValue);
  return sendCount;
}
static
PT_THREAD(run_analog(struct httpd_state *s, char *ptr))
{
  PSOCK_BEGIN(&s->sout);
  PSOCK_GENERATOR_SEND(&s->sout, generate_analog, s);
  PSOCK_END(&s->sout);
}
```

Esta metodología puede ser utilizada no solo para acceder a información generada desde el kit, sino también para indicar el tipo de acción a realizar obteniendo como resultado de la operación.
En lo que respecta al envío de información desde el navegador es necesario alterar una vez más la implementación del servidor web para que sea capaz de procesar peticiones GET donde se incorporen variables y valores junto al nombre del archivo solicitado. En este caso, la función encargada de procesar los flujos entrantes (handle_input), debe determinar la presencia de información a partir de la URL solicitada, extraerla y almacenarla como un campo de la estructura de datos asociada a la conexión. Para esto, es necesario modificar la estructura que representa la conexión agregando un nuevo campo para estos datos.
Por último, al momento de utilizar los datos enviados al servidor debe decodificarse esta información. Esto es necesario ya que al momento de remitir los datos desde el navegador, la información es codificada de modo de utilizar solo caracteres reservados codificando aquellos que no pueden ser directamente representados como valores hexadecimales.
Existen diferentes proyectos en los cuales se realizan las modificaciones mencionadas que fueron tomados como referencia para la implementación, entre ellos el proyecto uhttp-avr[1].
Utilizando estas nuevas modificaciones es posible implementar rutinas de generación de contenido dinámico que hagan uso de la información remitida, tal como se muestra en el listado siguiente.

```
unsigned char *parse_http_var(char **arg, char **k, char **v){
  unsigned char *ret=0;
  *k = (*arg);
  while((*(*arg) != 0) && (*(*arg) != '='))
    (*arg)++;
  if(*(*arg) == '=') {
    *(*arg)=0;
    *v = ((*arg)+1);
```

```
    (*arg)++;
    ret = 1;
  }
  while((*(*arg) != 0) && (*(*arg) != '&')) (*arg)++;
  if(*(*arg)=='&'){
    *(*arg)=0;
    (*arg)++;
  }
  return ret;
}

static unsigned short
generate_command(void *arg)
{
  char *k, *v;
  unsigned int char_send=0;
  unsigned char *str = ((struct httpd_state *)arg)->args;
  while(parse_http_var(&str, &k, &v)){
    char_send+= snprintf((char *)(char_send+uip_appdata),
                 UIP_APPDATA_SIZE-char_send,
                 "<entrada>\n<clave>%s</clave>\n", k);
    char_send+= snprintf((char *)(char_send+uip_appdata),
                 UIP_APPDATA_SIZE-char_send,
                 "<valor>%s</valor>\n</entrada>\n", v);
    if((strcmp(k, "led")==0) &&(strcmp(v, "on")==0))
      ledOn();
    else if((strcmp(k, "led")==0) &&(strcmp(v, "off")==0))
      ledOff();
    else if(!strcmp(k, "lcdTxt")){
      http_url_decode(v);
      printf("%s\n", v);
    }
    else if((strcmp(k, "lcdClear")==0)&&(strcmp(v, "clear")==0))
      _lcd_clear();
  }
  char_send+= snprintf((char *)(char_send+uip_appdata),
                UIP_APPDATA_SIZE-char_send,
                "</datos>\n");
  return char_send;
}

static
PT_THREAD(run_command(struct httpd_state *s, char *ptr))
{
  PSOCK_BEGIN(&s->sout);
```

```
  PSOCK_GENERATOR_SEND(&s->sout, generate_command, s);
  PSOCK_END(&s->sout);
}
```

Aquí el campo utilizado para el almacenamiento de los argumentos enviados (args) es analizado buscando pares clave/valor (rutina parse_http_var), donde el valor de la clave indica el tipo de operación a realizar y el valor constituye el argumento para el mismo. Notese que en el listado anterior, con el objetivo de reducir los requerimientos de memoria RAM, se utiliza el mismo campo de la estructura de conexión y dos punteros para identificar el comienzo de los pares clave (char *k) y valor (char *v), posteriormente se vuelve a utilizar el mismo campo para contener el valor de la clave decodificado tras invocar a la rutina http_url_decode. Esto es posible ya que el resultado de la decodificación contendrá la misma o menor cantidad de caracteres que la cadena codificada.
Notese también en la implementación anterior que según se identifican los argumentos asociados a la petición, se generan salidas que permiten comprobar su correcta interpretación desde cliente.

5. Aplicación cliente

La aplicación cliente presenta dos modos de interacción con el servidor, por un lado automáticamente a intervalos regulares solicita el envío de los valores asociados a las entradas y permite, según la acción del usuario, realizar operaciones específicas. Más allá del tipo de solicitud realizada, la mecánica utilizada es la misma; la diferencia es que en el primer caso la petición no requiere el uso de argumentos y en el segundo, según la acción deseada, se deben transmitir parámetros adicionales.
La implementación se basa entonces en el uso de instancias de la clase *URLLoader*, la cual permite permite realizar una petición desde una aplicación Flash y, una vez completada, dispara la ejecución de un evento en el cual es posible procesar la información recibida. Dado que ésta tiene estructura XML, es posible utilizar una instancia de la clase *XML* para acceder a los datos de interés.
Para lograr graficar los datos se implemento una película (MovieClip), que incorpora rutinas para el procesamiento de la información recibida y su representación. De este modo, creando 5 instancias de esta película es posible representar los valores obtenidos para cada sensor. Disparando la solicitud de las entradas repetidamente utilizando un reloj (instancia de la clase Timer) es posible lograr una gráfica dinámica que represente los valores obtenidos en el conversor. En el siguiente fragmento código se muestra la implementación correspondiente en ActionScript, este es ejecutado al momento de cargar la película principal.

```
var analogInput:XML;
var xmlLoader:URLLoader = new URLLoader();
var reloj:Timer = new Timer(2000, 0);
```

```
xmlLoader.addEventListener(Event.COMPLETE, actualizarAnalogIn);
reloj.addEventListener("timer", onTimer);
reloj.start();

this.grafica1.setMaxY(1024.0);
this.grafica2.setMaxY(1024.0);
this.grafica3.setMaxY(1024.0);
this.grafica4.setMaxY(1024.0);
this.graficaTemp.setMaxY(125.0);

function onTimer(event:TimerEvent):void{
  reloj.stop();
  xmlLoader.load(new URLRequest("analog.sxml"));
}

function actualizarAnalogIn(e:Event){
var temp:Number;
var tempIn: Number;
analogInput = new XML(e.target.data);
grafica1.actualizarDatos(parseInt(analogInput.entrada[0].valor));
grafica2.actualizarDatos(parseInt(analogInput.entrada[1].valor));
grafica3.actualizarDatos(parseInt(analogInput.entrada[2].valor));
grafica4.actualizarDatos(parseInt(analogInput.entrada[3].valor));
tempIn = parseInt(analogInput.entrada[4].valor);
temp = -(((((tempIn/1024.0)*3.3)-2.7)*75.0)+55.0);
graficaTemp.actualizarDatos(temp);
analogInput = null;
reloj.start();
}
```

Las acciones de control sobre el kit, debido a que requieren el uso de argumentos (por la forma de implementación) utilizan un objeto de la clase *URLVariable*, de este modo es posible enviar una petición GET al servidor web dando parámetros adicionales. En el siguiente fragmento de código se muestra la implementación de la rutina que permite enviar texto desde la aplicación cliente al servidor web para ser visualizada en el LCD. Este código es ejecutado en respuesta al evento click disparado por una película que cumple las veces de botón acción, implementaciones similares se encuentran asociadas a las diferentes acciones.

```
var request:URLRequest = new URLRequest("command.sxml");
var cmd:URLVariables;
var cmdLoader:URLLoader = new URLLoader();
cmdLoader.addEventListener(Event.COMPLETE, cmdFinalizado);
this.btnLCDSend.addEventListener(MouseEvent.CLICK, eventLCDSend);

function cmdFinalizado(e:Event){
var cmdResult:XML;
```

```
cmdResult = new XML(e.target.data);
this.labEstado.text = cmdResult.entrada.clave + "(" + cmdResult.entrada.valor + ")";
cmdResult = null;
}

function eventLCDSend(e: Event):void{
cmd = new URLVariables();
cmd.lcdTxt = this.txtLcd.text;
request.data = cmd;
cmdLoader.load(request);
}
```

En la figura 3 se muestra al aplicación en ejecución manteniendo conexión con el sistema embebido.

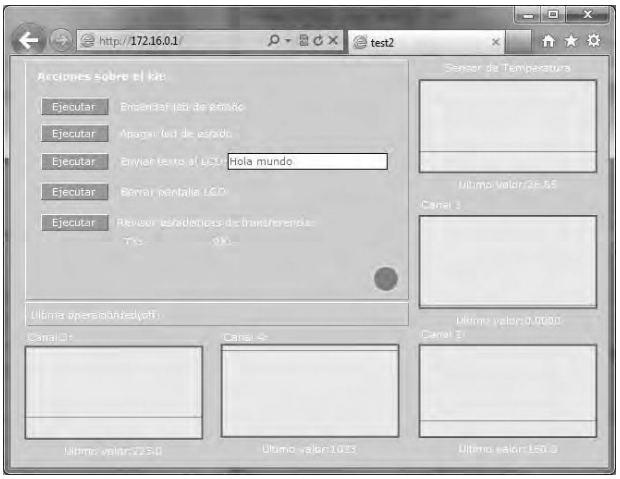

Figura 3. Aplicación final ejecutándose en un navegador y accediendo a la información del kit de desarrollo

6. Resultados y Conclusiones

El resultado logrado tanto desde el punto de vista del sistema embebido como desde la aplicación web fue exitoso. En lo que respecta al sistema embebido, el consumo de memoria fue bajo considerando el tipo de aplicación, rondando los 60 KB de memoria de programa (57 KB de código ejecutable y 3 KB de valores de inicialización) y 50 KB de RAM utilizando para su construcción el compilador GCC versión 4.5.2.
La arquitectura de la aplicación base utilizada como servidor web posibilitó un

desarrollo rápido de las modificaciones presentadas.
Se presentaron inconvenientes cuando el servidor recibía simultáneamente varias solicitudes desde la misma conexión, por ejemplo si estaba en proceso una petición para actualizar el valor de las entradas analógicas y se enviaba una acción sobre el kit. Esto fue resuelto implementando desde la aplicación cliente un mecanismo que solo permite la ejecución simultánea de una petición. Debido a esto se evaluó el funcionamiento cuando la aplicación cliente era ejecutada desde tres terminales no encontrándose ningún inconveniente.
En lo que respecta al ciclo de actualización de las gráficas, este no es periódico ni constante debido a que debe esperarse la respuesta del servidor web, la cual tendrá una velocidad dependiente de la carga de conexiones. Sin embargo, en aplicaciones como la presentada este no es un factor significativo.
La aplicación cliente requirió cierto cuidado en su desarrollo para evitar que su tamaño escalase. Debido a esto no se utilizaron controles de formulario, cuando se los intentó utilizar el tamaño de la aplicación llegó a 12 KB. Se utilizaron solo películas desarrolladas especialmente y cuadros de entrada de texto estándares. El tamaño total de la aplicación fue de 5 KB, teniendo el sitio completo un tamaño de 7 KB.
De los tamaños analizados podemos concluir que el uso de aplicaciones Flash como se ha presentado, permite mejorar ampliamente la experiencia de usuario no representando una carga para el sistema. Debido a que la comunicación desde el sistema embebido hacia la aplicación clientes se realizada utilizando XML y que este no incluye información de formato se logra un flujo de información mucho menor que cuando su utilizan páginas html dinámicas. Se tiene entonces una única aplicación encargada de la representación de los datos e interacción con el usuario y múltiples páginas dinámicas que generan pequeñas salidas de información dedicada. Esto repercute a su vez en el sistema embebido disminuyendo su carga de procesamiento y mejorando su rendimiento.
Un aspecto significativo de la implementación, es que al utilizar XML para realizar el intercambio de datos entre el sistema embebido y el cliente aparece un nuevo conjunto de posibilidades y aplicaciones debido al gran número de tecnologías que utilizan este formato para intercambiar información.
Un área por explorar es la posibilidad de implementación de Servicios Web (*Web Service*), tanto desde el sistema embebido así como teniéndolo como cliente, para lograr un mecanismo de interacción estándar con otras aplicaciones.

Referencias

1. micro httpd server based on modified uip-1.0 stack, running on 8bit avr processor and enc28j60, 2009. `http://code.google.com/p/uhttpd-avr/`.
2. Adam Dunkels. The uip embedded tcp/ip stack, 2006. `http://dunkels.com/adam/uip/`.
3. Texas Instruments. *Stellaris LM3S2776 Microcontroller Data Sheet (Rev. G)*, 2012. `http://www.ti.com/product/lm3s2776`.
4. JL Romkey. Rfc 1055: Nonstandard for transmission of ip datagrams over serial lines: Slip. *Chapter 11 Networking*, 1988. `http://www.ietf.org/rfc/rfc1055.txt`.

5. D. Stipanicev and J. Marasovic. Networked embedded greenhouse monitoring and control. In *Control Applications, 2003. CCA 2003. Proceedings of 2003 IEEE Conference on*, volume 2, pages 1350–1355. IEEE, 2003.