Revisión	Fecha	Comentario	Autor
0	02/09/2009	Los conceptos y herramientas de programación que serán utilizados para esta nota fueron oportunamente tratados en trabajos anteriores: - CoAN-007: *RabbitWeb, Javascript, CSS*. - CoAN-010: *RabbitWeb y AJAX*.	Ulises Bigliati

ÍNDICE

Aclaraciones previas

Sobre esta nota

El presente trabajo reúne una serie de elementos de programación web cuya utilización es libre, gratuita e independiente de la plataforma. La mayoría de aquellos lo son por tratarse de estándares y están simplemente disponibles en forma nativa en los navegadores web. Otros en cambio, lo son en virtud de que su fabricante los distribuye como versiones libres con ciertas condiciones. Tal es el caso del objeto central de este proyecto, que utilizaremos para la realización de los gráficos en tiempo real. Se trata de una animación Flash (archivo .SWF), cuyo desarrollador es **amCharts**[1] **(** http://www.amcharts.com/) quien nos impone como única condición para utilizar libremente y sin restricciones gran parte de sus productos, que el link hacia su sitio web se encuentre visible al desplegar el gráfico (lo cual no podremos evitar, a menos que el lector esté dispuesto a hacerlo clandestinamente).

[1] El fabricante se jacta (según consta en su sitio web) de contar entre sus clientes a las siguientes empresas: Microsoft, Sony, Cisco, NASA, Motorola, Procter and Gamble, Siemens, Merrill Lynch, ABN AMRO, ABB, NBC Universal, Société Générale, Syracuse University, y otras...

Conceptos previos y herramientas utilizadas

Tal como se adelantaba en los comentarios del encabezado, en este trabajo se emplean una serie de herramientas para el desarrollo en arquitectura web que el lector deberá conocer de antemano, al menos conceptualmente:

- Rabbitweb - Javascript / DOM - XML - HTTP – AJAX – CSS -

Cabe mencionar que esos elementos, en mayor o en menor medida fueron ya tratados y se han dado referencia de buenas fuentes para su aprendizaje en notas de aplicación anteriores, fundamentalmente CoAN-007 y CoAN-010.

Fuera de alcance

No es objeto de esta nota proporcionar detalles sobre los estándares de la arquitectura web o sobre metodologías de programación web, ni tampoco sobre tecnologías subyascentes (ej. networking).

Objetivos

El objetivo de este trabajo es proporcionar una herramienta más para el desarrollador de sistemas embebidos que requiera agregar funcionalidad web a sus diseños y en particular, cuando se trate de la confección de interfaces web de monitoreo y control.
El proyecto que se expone a continuación ofrece la posibilidad de generar gráficos en tiempo real a partir de valores proporcionados por nuestro equipo remoto con un mínimo costo en términos de transferencia de datos y procesamiento.
Si bien el proyecto está basado en el entorno de desarrollo de Rabbit, los elementos centrales de la aplicación son totalmente independientes de la plataforma.

Introducción

En este trabajo nos focalizamos en la implementación de una interfaz web para sistemas dedicados con la funcionalidad de monitoreo de una o mas variables numéricas, ya sean analógicas o digitales, pero como alternativa a la simple exposición del valor medido, ofrecemos la posibilidad de graficar en tiempo real el o los valores expuestos por el equipo remoto, pudiendo de esta forma visualizar la evolución de la señal en el tiempo, y no solo eso, sino que además, se podrán comparar las diferentes señales graficadas superponiendolas sobre el plano definido por el mismo par de ejes cartesianos.

Una gran ventaja del mecanismo que proponemos, es que nuestro servidor web, que en este caso se ejecuta sobre un módulo Rabbit, no estará al tanto de ninguno de los métodos que utilicemos, ya que toda la magia ocurre en el browser, del lado del cliente y nuestro servidor web se limitará simplemente a proveer los archivos que el browser solicite, entre ellos, el archivo *.SWF* que compila la animación web que genera los gráficos, y un pequeño archivo ASCII en formato *XML* que transportará los datos de las variables.
En virtud de lo expuesto en el párrafo precedente, afirmamos que el material desarrollado en el presente proyecto es independiente de la plataforma.

Requerimientos de hardware y software

Para la puesta en marcha de este proyecto utilizamos un módulo Rabbit MiniCore RCM5700 en su placa de prototipos, y de tal forma contamos con un microprocesador Rabbit® 5000 funcionando a 50MHz, interfaz Ethernet 10/100Base-T, memoria flash de 1 MB, y SRAM de 128K. El motivo de haber seleccionado este módulo es fundamentalmente el testeo de dicho hardware, lo cual significa que cualquier otro módulo Rabbit con interfaz Ethernet podría servir para probar este desarrollo.

En cuanto al software, se utilizó el *Dynamic C10.46* para desarrollar el firmware y grabar el módulo y durante la mayor parte del proyecto fue utilizado para las pruebas el navegador *Mozilla Firefox 3.5.2*. En pro de la portabilidad, también se verificó el funcionamiento de la aplicación utilizando otros exploradores que no vale la pena mencionar.

El proyecto propuesto

El software

El programa que desarrollamos para este proyecto tiene la misión de demostrar una metodología mediante la cual podremos realizar el monitoreo gráfico vía web de diferentes señales generadas o adquiridas por un equipo remoto, en este caso, utilizando módulos *Rabbit* junto con su servidor *HTTP* embebido sumándole las facilidades que nos otorga *Rabbitweb*.
Para esto, hemos construido un esquema basado en tres aspectos fundamentales que se comentan brevemente a continuación:

1) La simulación de las señales que serán monitoreadas a través de la web.

Para esto realizamos un típico programa basado en un bucle infinito, dentro del cual se define una estructura de `costates` desde las cuales generamos la simulación de las diferentes señales que son teóricamente captadas periódicamente desde sus correspondientes canales analógicos.
Los valores "leidos" de estos canales virtuales son almacenados en cada instante, a modo de instantánea, en un vector que siempre posee los valores de la última lectura.

2) La implementación del mecanismo necesario para poder proporcionarle a uno o mas clientes web (browsers) los datos provenientes de dichas señales o canales.

Para lograr este mecanismo nos valemos por supuesto del *HTTP* server de *Rabbit* y de su lenguaje de scripting del lado de servidor, el *Rabbitweb.*
Utilizamos estos elementos para la generación de un archivo *XML* que cumplirá con la función de transporte de los datos de los canales virtualizados en el punto anterior. Este archivo es generado y enviado por el sevidor web solo como respuesta a una petición *HTTP* emitida por un web browser, y los valores devueltos en este archivo *XML* son los extraidos del vector mencionado en el punto (1).
Todo esto se materializa simplemente en un archivo ASCII que contiene el formato *XML* y a su vez lleva embebido el código de scripting *Rabbitweb* para extraer los valores generados en el punto previo.

3) El desarrollo del esquema que nos permita solicitar, recibir, procesar y finalmente generar los gráficos que serán despegados en la pantalla del navegador.

Esto se logra gracias a un conjunto de elementos web, que serán almacenados y suministrados por el servidor *HTTP* de *Rabbit*, estos elementos se materializan en diversos archivos:

- Archivos Javascript (.*JS*) conteniendo diversas utilidades.
- El archivo flash (.*SWF*), núcleo de la aplicación y responsable de generar la animación que dibujará los gráficos a partir de los datos suministrados por el archivo *XML*.
- La página web principal (.*HTML*) que contiene además el código Javascript embebido, responsable del funcionamiento de todo el mecanismo buscado en este punto.
- Las reglas de estilo (.*CSS*) que proporcionan la estética de la interfaz.
- Y aquí tambien podríamos volver a referirnos al archivo de datos (.*XML*) ya que funcionalmente los puntos (2) y (3) están solapados.

Todos estos elementos (con excepción del archivo flash (.*SWF*) que ya está compilado), poseen código interpretado por el navegador web, y los hemos diseñado, escrito y editado con total independencia del *Dynamic C*, ya que se trata de estándares de la arquitectura web que son totalmente independientes de la plataforma[2] que los contenga y/o proporcione. Por ello, el concepto de esta aplicación es perfectamente portable a cualquier otra plataforma.
En virtud de lo expuesto, la estructura que adquiere este proyecto en cuanto a su organización de archivos se refiere, puede apreciarse a continuación:

[2] Aquí nos referimos a la configuración de hardware-software que proporcione el soporte necesario para que un cliente web pueda ejecutar con éxito la aplicación. En resúmen, un servicio *HTTP* mas un mecanismo responsable de suministrar datos dinámicamente actualizados dentro del archivo *XML*.

myProyects___CoAN-013

```
|___CoAN-013.c
|___CoAN-013.dcp
|___lib.dir
|___lib
|       |___datetime.lib
|       |___iweb.lib
|___iweb
        |___index.html
        |___style.css
        |___values.xml
        |___utils.js
        |___menu.js
        |___chart
                |___amline.swf
                |___swfobject.js
```

Archivos de configuración del proyecto.

Archivos escritos y compilados con Dynamic C.

Archivos escritos con editores de texto. Son luego cargados en memoria flash del módulo por el Dynamic C (directiva #ximport) su contenido solo es importante para el browser.

Archivos que implementan la generación de los gráficos en tiempo real. Solo interesan al browser y también son cargados por el DC en memoria flash del módulo.

Módulos del proyecto

A continuación se dará una breve explicación general sobre cada uno de los módulos del proyecto y cuando el contenido de aquel lo amerite se ahondará en mayores detalles sobre el código fuente involucrado y su funcionamiento específico.
Con fines de no atentar contra la claridad y concisión de esta nota no se transcribirán partes del código fuente que no merezcan ser particularmente analizadas, ya que en algunos casos el código se auto-explica y en otros se encuentra convenientemente comentado.

Módulo CoAN-013.c

Este archivo fuente contiene el programa principal, es el bloque de código desde el cual invocamos al resto de los módulos y definimos una serie de parámetros necesarios para que nuestra aplicación funcione como queremos además de realizar alguna otras tarea muy sencillas. Aquí vamos a encontrar las siguientes secciones:
- Definiciones de los parámetros de networking bajo el comentario:

```
/*************************************
*        NETWORK CONFIGURATION          *
*************************************/
```

- Invocación de library's de TCP/IP y del servicio de HTTP y *Rabbitweb* bajo el comentario:

```
/*************************************
*  Networking modules invocation      *
*************************************/
```

- Invocación de library's de usuario bajo el comentario:

```
/*************************************
*  application modules for this project  *
*************************************/
```

- Declaración de variables vitales (y otras no tanto) para la generación de gráficos desde la página web bajo el comentario:.

```
/*************************************
*  variables for samples simulations     *
*************************************/
```

- Finalmente el cuerpo del programa, desde el cual simulamos lógica y/o matemáticamente los valores que luego serán tomados desde la página web, por supuesto dentro de la función `main()`.

Módulo datetime.lib

Este módulo solo presenta funciones de soporte y ya se ha utilizado trabajos anteriores, no ahondaremos en detalles al respecto.

Módulo iweb.lib

El módulo `iweb.lib` también es ampliamente utilizado en nuestros proyectos, dentro de sus entrañas nos encargamos de preparar el terreno para que el servidor web pueda disponer de todos los elementos

necesarios para proveerle al web browser que visitará el sitio alojado en nuestro módulo *Rabbit* todo lo necesario para que la aplicación propuesta funcione, no nos extenderemos en mayores comentarios al respecto dado que el código fuente de este módulo es muy similar al utilizado y explicado en la nota de aplicación CoAN-010.
Solo cabe destacar que estamos incluyendo un nuevo elemento que hasta el momento no habíamos utilizado en nuestros proyectos, esto es, el archivo Flash (.swf). Dentro de la tabla de recursos se ve así:

```
SSPEC_RESOURCE_XMEMFILE("/chart/amline.swf",    amline_swf  ),
```

En función de esto debemos declarar un nuevo tipo MIME en la tabla correspondiente, de esta forma:

```
SSPEC_MIME(".swf", "application/x-shockwave-flash"),
```

Por lo demás, no hay nada nuevo, pero la inclusión de un nombre de tipo *MIME* tan largo nos obligó a modificar el tamaño máximo para dichos nombres mediante estas definiciones (esto es en *CoAN-013.c*):

```
#define SSPEC_MAXNAME     30
#define HTTP_MAXNAME      SSPEC_MAXNAME
```

El subdirectorio iweb

En este directorio, que es también muy habitual en nuestros proyectos, reunimos a todos los elementos que conforman nuestra aplicación web. Dado que una vez mas, estamos en presencia de la reutilización de muchos de estos objetos solo haremos un breve repaso de ellos deteniendonos solo en los aspectos más relevantes.

La hoja de estilos “style.css”

Ya la hemos utilizado en notas previas (CoAN-007, CoAN-010) y aquí solamente agregamos y/o modificamos algunas reglas de estilo para lograr la apariencia deseada de la interfaz de nuestra aplicación. Cabe mencionar que la forma de vincular la hoja de estilos con la página web es mediante la siguiente referencia ubicada dentro del `<head >`:

```
<link rel="stylesheet" type="text/css" href="style.css" />
```

Archivo *Javascript* “utils.js”

Se trata de un archivo de código fuente interpretable por el browser. Es realmente una biblioteca de funciones (o library), en este caso en particular contiene utilidades generales:

```
checkTime(), getTime(), newAjax(), isNatural(), isInteger(), isNumeric()
```

Aquí nos interesa mencionar la forma en que este archivo es invocado por el browser, esto es, incluyendo la siguiente línea de código en la página web que deba utilizar las funciones de esta biblioteca, normalmente pondremos este tipo de referencias entre los tags `<head>` y `</head>`.

```
<script language ="javascript" type ="text/javascript" src="utils.js" ></script>
```

Archivo *Javascript* “menu.js”

Este archivo *Javascript* proporciona una funcionalidad que bien podríamos llamar superflua, pues no guarda ninguna relación con la aplicación en sí, por lo tanto en favor de la claridad no daremos mas detalles al respecto dejando este asunto para otras notas de aplicación. Solo agregamos que, tal como puede intuirse, este archivo da soporte a la implementación de un menú desplegable sencillo y versátil.

Archivo *XML* “values.xml”

Este es el archivo de transporte de datos, y es vital para la aplicación ya que llevará consigo los valores que alimentarán los gráficos que se desplegarán en la interfaz web del cliente (browser). Los datos se utilizarán en forma de par ordenado: el tiempo para el eje “x” (contenido del elemento `<dt></dt>`), y el valor de los diferentes canales para el eje “y” (contenido de los elementos `<ch></ch>`), luego con cada canal se podrá conformar un gráfico diferente, aunque superpuestos en un mismo plano definido por un único par de ejes.
El archivo “values.xml”, se hizo famoso en la nota de aplicación CoAN-010 donde explicamos una forma para utilizar *AJAX* y dado que aquí lo utilizamos de manera análoga, los conceptos listados a continuación fueron cubiertos en dicha nota y no lo repetiremos aquí, a saber:

- La estructura del archivo (formato *XML*)
- El origen de los datos que transporta (variables `#web`) y la forma de obtenerlos (*Rabbitweb*)
- Cómo es solicitado y recibido por la aplicación web del lado del browser (*AJAX*)
- Cómo se extraen y se consumen los datos en el browser (*Javascript*, *DOM*).

Página principal "index.html"

En este archivo se construye la página web principal de la aplicación, cuando dirijamos nuestro browser a la dirección IP de nuestro módulo *Rabbit*, estaremos solicitándolo. En esta página se insertarán según corresponda los diferentes objetos web del proyecto, todos ellos contenidos físicamente por el directorio "iweb" y el sub-directorio "chart".
Volveremos a este módulo más adelante, ya que por su complejidad debe ser tratado más extensamente.

El subdirectorio "chart"

Creamos dentro del proyecto un subdirectorio dentro del cual colocamos los dos archivos que nos permiten lograr la presentación de los gráficos en tiempo real. Este subdirectorio lo creamos simplemente con fines organizativos.

El Archivo "swfobject.js"

Este archivo implementa el objeto *SWFObject*[3], que expone una *API* con unas pocas funciones de interfaz que simplifican al máximo las operaciones de inserción y manipulación de contenido *Adobe Flash* en páginas web. El script detecta el plug-in de *Flash* y su versión en la mayoría de los browsers y evita la necesidad de tener que conocer los detalles de parametrización que de otra forma deberíamos manejar. Veamos cómo se utiliza:

Primero: incluimos el archivo *Javascript* en la página web como de costumbre:

```
<head >[...]
<script language ="javascript" type ="text/javascript" src="chart/swfobject.js"></script>
[...]</head>
```

Segundo: en alguna parte del `<body>` debemos incluir un contenedor dentro del cual el objeto *SWFObject* va a escribir el script requerido para incrustar el contenido *Adobe Flash* en la página. Normalmente haremos esto con un `<div>` al cual se le debe asignar un ID:

```
<div id="flashcontent0" >
      Este texto será reemplazado por el chart
</div>
```

Tercero: dentro de un bloque de script[4] creamos el objeto *SWFObject*, que será el encargado de facilitarnos la tarea de embeber el objeto *Flash* en la página web y estableceremos así una referencia al contenido que nos permitirá manipularlo más adelante:

```
var chart = new SWFObject("chart/amline.swf", //Ruta de acceso a la animación compilada
                          "amline",           //ID de referencia para Javascript
                          "720",              //Ancho de la animación
                          "400",              //Altura de la animación
                          "8",                //Versión del reproductor requerida
                          "#FFFFFF"           //Color de fondo.
                          );
```

Cuarto: a continuación, con la siguiente línea:

```
chart.write( id )
```

Se le da la orden al objeto *SWFObject* de volcar en el contenedor referenciado por el *id* (el `<div>` del segundo paso) el código necesario para desplegar el contenido *Adobe Flash*, en este caso será nuestro chart que está compilado en el archivo *amline.swf*.

Opcional: el objeto *SWFObject* expone además la función `addVariable()` mediante la cual es posible manipular variables dentro del objeto *Flash* compilado, modificando su comportamiento:

---

[3] Documentación al respecto: http://blog.deconcept.com/swfobject/ y http://code.google.com/p/swfobject/

[4]
```
<script type ="text/javascript" >
      [ código fuente Javascript ]
</script>
```

```
chart.addVariable("variable1", valor1 );
```
Utilizando antes esta función antes de `.write()` podemos modificar nuestro contenido *Flash* interactivamente de diversas maneras durante el uso de la aplicación desde nuestro browser.

El componente "amline.swf"

Este es el archivo con contenido *Adobe Flash* compilado, se trata de la aplicación gráfica que nos va a permitir desplegar los gráficos en tiempo real alimentandose de datos en formato *XML*. Por su parte, este objeto también expone una serie de funciones de interfaz[5] para su manipulación, de las cuales solo nos interesa para nuestro proyecto la siguiente:

```
flashMovie.appendData(data[,remove_count])
```

Con ella podemos agregar nuevos valores al conjunto de datos existente en el gráfico sin necesidad de recargar el gráfico completo. Los datos se pasan por el parámetro `data` y se pueden conformar en formato *CSV* o *XML*, nosotros elegimos este último. El parámetro `remove_count` es opcional e indica el número de "columnas" de datos que deben removerse desde el origen del gráfico. En este proyecto nosotros hacemos algo similar a esto:

```
data =
"<chart>" +
      "<series>" +
            "<value xid='" + sn + "'>" + dt + "</value>" +
      "</series>" +
      "<graphs>" +
            "<graph gid='" + chid +  "'>" +
                  "<value xid='" + sn + "'>" + chval + "</value>" +
            "</graph>" +
      "</graphs>" +
"</chart>";

flashMovie.appendData( data );
```

Donde:

- `sn` es un número de serie;
- `dt` contiene fecha y hora, extraídas de del archivo "values.xml"
- `chid` es el id del gráfico que estamos generando, pues podemos incluir mas de uno entre las etiquetas `"<graphs></graphs>"`. En el contexto de nuestra aplicación `chid` es el número de canal (virtual) de la señal que estamos simulando.
- `chval` es el valor "medido" de la señal simulada, el cual es extraido, igual que `dt`, del archivo "values.xml".

Finalmente resta mencionar que además de la interfaz para la manipulación dinámica de los gráficos, disponemos de una serie de variables de parametrización del chart. Dichas variables son accesibles gracias a la función `.addVariable()` descripta más arriba, que es provista por el objeto *Open Source SWFObject*.

Una de las variables que nos interesa modificar es `"chart_settings",` como su nombre lo indica podemos modificar configuraciones del gráfico: color, etiquetas, justificación, etc. y lo hacemos de esta forma:

```
var settings = "<settings><background><color>#FFFFFF,#9E9EFF</color><alpha>100</alpha>"
+ /* [... más configuraciones ...] */ + "</settings>";
chart.addVariable("chart_settings", encodeURIComponent[6](settings));
```

[5] Documentación al respecto: http://www.amcharts.com/docs/v.1/bundle/javascript/function_list

[6] `encodeURIComponent()` es una función *Javascript* nativa en los navegadores que codifica un *string* reemplazando caracteres no permitidos por su equivalente código de escape. Mas datos en http://www.w3schools.com/jsref/jsref_encodeURIComponent.asp

Donde `settings` como puede verse, es una variable que contiene un string formateado en *XML* llevando consigo las configuraciones que estaríamos modificando.

También cabe destacar el parámeto `"chart_data"` igualmente accesible mediante la función:

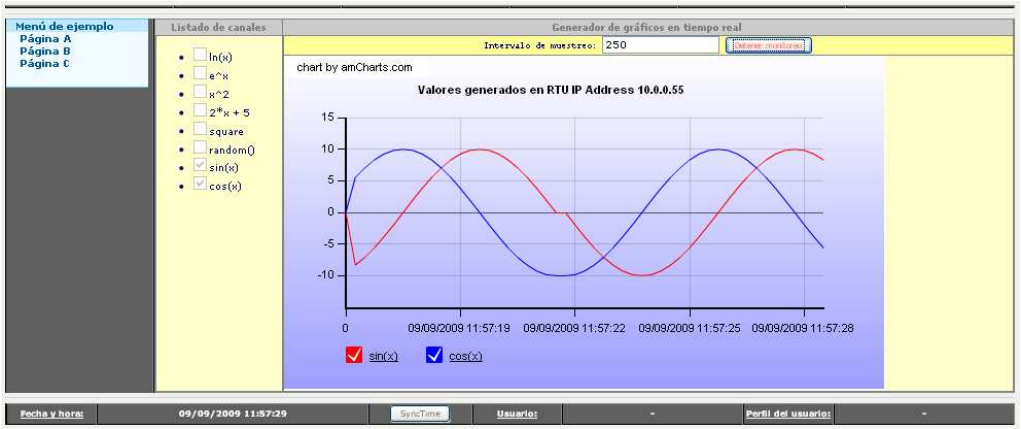
```
chart.addVariable("chart_data", encodeURIComponent(graphs) );
```

Esta función la utilizamos para ingresar datos a un gráfico directamente desde su página web, podríamos decir, en forma estática, ya que cargaríamos un conjunto de valores completo cada vez. El parámetro `graphs` contiene el set de datos iniciales que se están cargando en el gráfico, y nosotros lo utilizamos con fines de inicialización, de manera tal de preparar el o los gráficos con un conjunto vacío de datos. Luego se irían actualizando los gráficos con la función `.appendData()` cada vez que lleguen nuevos valores.
El formato del string contenido en la variable `graphs` es el mismo utilizado en la variable `data` pasado a la función `.appendData()`.

Se puede encontrar mayor información al respecto de estas parametrizaciones y de muchas otras que no hemos utilizado, en el sitio web del fabricante, sección de documentación:
http://www.amcharts.com/docs/v.1/bundle
http://www.amcharts.com/docs/v.1/bundle/basics/adding_chart_to_a_page

La aplicación: reuniendo todas las partes.

La interfaz gráfica de la aplicación, muestra en funcionamiento algo similar a esto (solo que la señal se apreciará en movimiento y en tiempo real):



Esta vista corresponde a la visualización de la página principal “index.html”, así que volveremos a este módulo para completar la descripción del código fuente de este trabajo. Veremos a continuación como se terminan de ensamblar todas las piezas, por supuesto, reduciendo el código presentado en estas páginas a los fragmentos más significativos, el lector podrá luego contrastar lo aquí descripto consultando el código fuente completo que acompaña a esta nota.

La interfaz de usuario

Sintéticamente, diremos que el usuario tiene la posibilidad de seleccionar una o más cajas de verificación que refieren a un canal simulado diferente, y además de indicar en una caja de texto, el intervalo de consulta y presionando el botón adyascente se dispara la presentación de los gráficos conforme a los datos recibidos del equipo remoto.
A continuación vemos el código de *Rabbitweb* embebido en el *HTML* donde generamos dinámicamente del lado del servidor (módulo *Rabbit*) la lista de *checkboxes* en base a la descripción de los canales que guardamos en un array de caracteres (`chanDesc[]`) que se encuentra en el archivo principal del proyecto de *Dynamic C* (CoAN-013.C):

```
<ul >[7]
<?z for ($A = 0; $A < 8; $A++) { ?>
   <li><input id="<?z print($A) ?>" type="checkbox" value="<?z print(@chanDesc[$A]) ?>" name="ch" />
        <?z print( @chanDesc[$A] ) ?>
   </li>
<?z } ?>
</ul>
```

Con esto tendremos entonces la lista de casillas de verificación que están vinculadas con los canales virtuales. Por otro lado la línea siguiente:

```
<input type="button" onclick="setTitle(this);" value="Iniciar monitoreo"
class="dataButton" />
```

es el código *HTML* que define al botón que dispara los gráficos, lo que debemos notar es la llamada a la función `setTitle(this)` ante la presión del botón.

La función setTitle()[8]

Esta función recibe como parámetro la referencia al objeto que la invoca y como una tarea accesoria le modifica el texto según el estado del monitoreo (del tipo on/off). Sin embargo su tarea primordial es inicializar o detener el llamado recurrente de la función *AJAX* que veremos luego, e inicializar el gráfico llamando a la función `chartsRTInit()`.
El disparador del mecanismo *AJAX* será el llamado inicial a la función `getRealtimeData( null )` junto con el establecimiento de la variable gloabal `ajaxstart` en `true`. Para detenerlo, bastará con establecer `ajaxstart` en `false`.

```
function setTitle(obj)
{
            ajaxstart = !ajaxstart;
            var chs = document.getElementsByName ("ch");

            if(ajaxstart){
                  chartsRTInit();
                  getRealtimeData( null );
                  obj.value = "Detener monitoreo"
                  obj.style.color = "red";
            }
            else{
                  obj.value = "iniciar monitoreo"
                  obj.style.color = "green";
            }
            for(var i=0; i< chs.length; i++)
                  chs[i].disabled= ajaxstart;
}
```

La función chartsRTInit()

Decíamos bajo el subtítulo anterior, que la función `setTitle()` llamará a esta función para la inicialización de los gráficos. Esto se realiza para preparar el terreno para la incorporación posterior y periódica de los valores de las señales correspondientes a los canales seleccionados por el usuario. A continuación podemos apreciar la definición de esta función, a la cual le hemos omitido las partes menos significativas del código, nótese que al comienzo del bloque de script, están las declaraciones de variables globales, y lo más notorio, la instanciación del objeto `SWFObject`, tal como comentábamos más arriba.

[7] Se omitió parte del código por claridad.
[8] Debemos reconocer que el nombre de esta función no refleja con mucha precisión las tareas que realiza.

```
<script type ="text/javascript" >
      var timer, ajaxstart=false;
      var poll_interval=250;
      var chart;
      chart = new SWFObject("chart/amline.swf", "amline", "720", "400", "8", "#FFFFFF");

            function chartsRTInit( )
            {
                  var chs = document.getElementsByName ("ch");
                  var graphs = "<chart> [...]<graphs>";
                  for(var i=0;i<chs.length;i++){
                        if( chs[i].checked )
                              graphs += "<graph gid='" + chs[i].id + [...];
                  }
                  graphs +="</graphs></chart>";

                  var settings = "<settings> [...] </settings>";
                  chart.addVariable("chart_settings", encodeURIComponent(settings));
                  chart.addVariable("chart_data", encodeURIComponent(graphs));
                  chart.write("flashcontent0");
            }
      [...]
      [...]

</script>
```

Conforme a los canales seleccionados, se van construyendo los conjuntos de datos necesarios para iniciar los gráficos en la variable graphs

-Parámetros
-Set de datos
-Volcado del contenido

Detrás de escena

La "función *AJAX*": getRealtimeData()

Luego de crear el objeto *Flash* y de inicializar los gráficos mediante la función precedente, se sucede la llamada a la función `getRealtimeData()` que implementa los mecanismos definidos por la metodología *AJAX*, los cuales fueron abordados en detalle en la nota CoAN-010.
Aquí se utilizan conceptualmente de la misma manera, siendo la única diferencia el destino final de los datos extraídos del archivo "values.xml", lo cual analizamos más adelante. Como puede verse, la estructura de la función *AJAX* es la clásica:

```
var sn=0; //Variable global usada como número de serie
function getRealtimeData( syncTime )
{
 var xmlHttp = newAjax();
 xmlHttp.onreadystatechange  =
 function()
 {
       if(xmlHttp.readyState==4)
       {

          Recepción del archivo XML con los datos (values.xml)

       }
 }
   Si syncTime está definido, estoy poniendo en hora el sistema, entonces se arma el parámetro adecuado
xmlHttp.open("POST","values.xml",true);
var params = (syncTime)? "strDateTime=" + getTime(): "timeRefresh=1";
xmlHttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xmlHttp.setRequestHeader("Content-length", params.length);
xmlHttp.send(params);

if(ajaxstart)
       timer=setTimeout("getRealtimeData(" + syncTime + ")",poll_interval);
else
       clearTimeout(timer);
}
```

Solicitud del archivo XML

Auto-llamado o fin del monitoreo.

Este es el intervalo de actualización que sale de la caja de texto en la interfaz de usuario

A continuación analizamos lo que sucede al recibir los datos desde nuestro servidor web, es decir los valores de las señales que estamos simulando contenidos en el archivo XML:

Guardamos los valores de elementos "ch" del archivo values.xml en un array y recuperamos el time-stamp

```
var xmlValues  = xmlHttp.responseXML.documentElement.getElementsByTagName("ch");
var dt =
xmlHttp.responseXML.documentElement.getElementsByTagName("dt")[0].childNodes[0].nodeValue;
document.getElementById("stbDt").innerHTML = dt;
```

Obtenemos los checkboxes en otro array, para luego saber que canales están seleccionados y así graficarlos.

```
var chs = document.getElementsByName("ch");
```

Ya tenemos los valores para agregar al gráfico. También sabemos que canales seleccionó el usuario. Lo que sigue, es generar el string *XML* para actualizar los gráficos de los canales seleccionados mediante la función `appendData()`. Esto lo hacemos con una máquina de estado sencilla.

```
genNewChartData(0, 0,  sn, dt, STATE.BEGIN  );//state machine [init]
for(var i=0; i<chs.length; i++)
{
        if( chs[i].checked )//Si el canal está seleccionado:
        {
                genNewChartData(i, 0, 0, 0,  STATE.BEGINGRAPH );//state machine [new graph]
                genNewChartData(i, xmlValues[i].childNodes[0].nodeValue,sn,0,STATE.NEWVALUE );
                genNewChartData(0, 0,  0,  0, STATE.ENDGRAPH );//state machine [end graph]
        }
}
```

Al terminar de recorrer los canales, obtenemos el string completo en la variable `newSample`.

```
var newSample = genNewChartData(0, 0, 0,  0, STATE.END );//state machine [final]
chartRTUpdate( newSample );//update chart
sn++;
```

Al finalizar, actualizamos los gráficos, e incrementamos la cuenta de muestras (`sn++;`)

Antes de echarle un vistazo a la máquina de estados que usamos para generar el string *XML* con el cual inyectamos los datos recientes en el gráfico veamos la función `chartRTUpdate()` que no es más que una envoltura de la función `appendData()`. Solamente se agrega la obtención de una referencia al chart mediante su *ID* (especificado al crear el objeto *Flash* mediante el *SWFObject*):

```
function chartRTUpdate( newSerie  )
{
        var flashMovie = document.getElementById ("amline");
        flashMovie.appendData( newSerie );
}
```

La función genNewChartData()

Lo único que resta es simplemente observar la función utilizada para conformar el string *XML* que contiene los conjuntos de datos de los diferentes gráficos que estemos generando. Para esto realizamos una sencilla máquina de estados en la cual el estado es manejado por la funcion que llama, necesitamos además una variable global para ir conformando el string que se devolverá al "llamador" al finalizar las operaciones (estado `STATE.END`) en el resto de los estados, la función devolverá `null`.

```
var newSerie; //Variable global para formar el string XML
var STATE={BEGIN:-1, BEGINGRAPH:-2, NEWVALUE:-3, ENDGRAPH:-4, END:-5 }; // Definición de estados
function genNewChartData(chid,chval, sn, dt, st )
{
	switch(st)
	{
		case STATE.BEGIN: //init state
			newSerie= "<chart><series><value xid='" + sn + "'>" + dt +
			"</value></series><graphs>";
			break;
		case STATE.BEGINGRAPH: //new graph state
			newSerie+="<graph gid='" + chid +  "'>";
			break;
		case STATE.ENDGRAPH: //end graph state
			newSerie+="</graph>";
			break;
		case STATE.END: //end state
			newSerie += "</graphs></chart>";
			return newSerie;
		case STATE.NEWVALUE: //new data item
			newSerie +="<value xid='" + sn + "'>" + chval + "</value>";
			break;
		default:
			break;
	}
	return null;
}
```

`chid:` es el identificador del gráfico y se obtiene directamente del *ID* del *checkbox* correspondiente de esta forma se vincula unívocamente un gráfico con un checkbox y este con un canal virtual del equipo remoto. Solo tiene significado en el estado `STATE.BEGINGRAPH`

`chval:` es el valor actual que presenta la señal que se está graficando y se obtiene del archivo *XML* values.xml. Solo tiene significado en el estado `STATE.NEWVALUE`

`sn:` es un número de serie que identifica la posición actual en la serie numérica del gráfico. Solo tiene significado en el estado `STATE.NEWVALUE` y `STATE.BEGIN`

`dt:` es el valor actual de tiempo que se grafica en el eje x. Solo tiene significado en el estado `STATE.BEGIN`.

`st:` es el estado actual del proceso. Y debe tomar alguno de los siguientes valores: `STATE.BEGIN (-1), STATE.BEGINGRAPH (-2), STATE.NEWVALUE (-3), STATE.ENDGRAPH (-4)` o `STATE.END (-5)`

Consideraciones finales

Luego de plantear este trabajo creemos que esta propuesta constituye una buena alternativa en térmios de costos y de beneficios obtenidos a la hora de realizar aplicaciones de monitoreo y control basados en web. Los componentes de software utilizados pueden ser fácilmente incorporados a cualquier diseño embebido preexistente, ya que son independientes de la plataforma y solo requieren ser despachados por un servidor HTTP, mientras que todo lo demás sucede en el browser.
Por otra parte son de uso lilbre y gratuito, y están soportados por la mayoría de los navegadores lo cual garantiza la portabilidad.
Los tiempos de respuesta deberían ser evaluados teniendo en cuenta diversos factores (carga de la red, tiempos de procesamiento en cliente y servidor, etc.), pero lo que puede asegurarse es que la metodología usada se adapta para lograr mínimas transferencias de datos entre el equipo remoto y el browser, de forma tal que su desempeño en sistemas de tiempo real (dependiendo de lo críticos que sean sus procesos) puede llegar a ser muy aceptable y es muy viable su implementación.
Como contrapartida, podemos mencionar la dificultad que puede suponer la necesidad de conocer los distintos lenguajes de programación implicados: *C*, *HTML*, *Javascript*, *Rabbitweb*, *XML* y los diversos métodos de programación (*AJAX*) y modelos de objetos de soporte (*DOM*, *SWFObject*) y finalmente el producto de terceras partes que se está utilizando (*amline.swf* de *amCharts*).