EN ESTE NÚMERO REALIZAMOS UN ANÁLISIS DEL IBM VISUALAGE FOR JAVA 3.0

PROGRAMACIÓN ACTUAL

www.prensatecnica.com AÑO 3 • NÚMERO 34 995 pts.
PORTUGAL 995 ESC (CONT) 5,98 €

Protocolo WAP

Desarrollo ASP para telefonía móvil

En Silicio
ARQUITECTURA INTERNA DEL ATHLON K7

Seguridad
PROTOCOLO SECURE SOCKETS LAYER (SSL)

Windows NT
SERVICIO DHCP

Delphi
CÓMO ENCAPSULAR LA FUNCIONALIDAD DE UN ARCHIVO

Bases de Datos
EL PROGRESS 4GL

Espacio GNU
X-WINDOW: EDICIÓN DE TEXTO EN XLIB
PROGRAMACIÓN DE PÁGINAS WEB INTERACTIVAS CON PERL

Nivel Básico
TIPOS DE DATOS EN JAVA

En Vanguardia
XML: DOM (DOCUMENT OBJECT MODEL)

Contenido CD-Rom

DEMO:
•IBM VISUAL AGE FOR JAVA 3.0

SHAREWARE:
•Awawe 32

INTERNET
•GetRight
•Internet loafer
•BuddyPhone

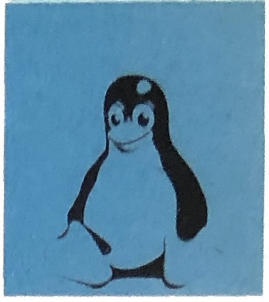
MULTIMEDIA
•Paint Shop Pro
•Reproductores White Tiger

PROGRAMACIÓN
•Diamond Access 1.7
•WinDriver CE 4.14

Construcción de DLLs en Visual Basic

Autor: Santiago Romero
santiago.romero @ iname.com

X-WINDOW



Edición de texto en XLib

En esta entrega final veremos cómo se puede entrar y editar texto. Además mostraremos una manera muy sencilla de realizar botones para nuestras ventanas, aprovechando el sistema de eventos, con lo que los botones se gestionarán a sí mismos, lo que dotará a nuestro programa de mucha sencillez a la hora de programar.

En esta última andadura en nuestro curso de XLib, veremos una sencilla manera de utilizar la edición de texto y botones, sin necesidad de incluir ni aprender funciones de Toolkits para X-Window. Los *toolkits* suelen proporcionar funciones ya preparadas para editar texto, pulsadores, tablas, botones, cajas de selección, etc. Si nuestra aplicación va a hacer un sencillo uso de las pulsaciones sobre botones o si sólo necesita entrar texto en ciertas ocasiones (introducción del nombre para *records*, petición de texto simple, etc.), podemos suplir el uso de Toolkits mediante las funciones que XLib nos proporciona.

EDICIÓN DE CADENAS DE TEXTO

Una de las necesidades básicas de cualquier programa suele ser la entrada de texto por parte del usuario. Una cosa tan sencilla en un ambiente como MSDOS, puede complicarse bastante en un entorno multitarea y de ventanas como X-Window (o Windows), sin el uso de *toolkits* apropiados ya preparados. Mediante las funciones básicas de XLib (es decir, sin el uso de *widgets* de entrada de texto de los más famosos *toolkits*, como GTK o QT), para realizar la entrada de cadenas de texto será necesaria la creación de una función que acepte como parámetros una cadena, el *offset* actual dentro de la cadena (la posición del cursor virtual de escritura) y la tecla pulsada, y que se encargue de la inserción, borrado y dibujado de caracteres en el vector y en la pantalla.
Un ejemplo de función de este tipo lo podemos ver en el **Listado 1** que se incluye en el CD-Rom que acompaña la revista, donde hemos creado una función llamada *EditaTexto* que acepta los siguientes parámetros:

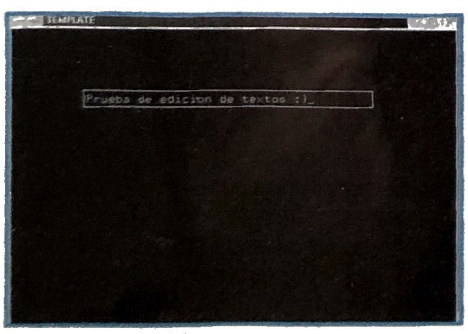


Figura 1. Salida del programa.

```
int EditaTexto( Display *display, Window
window, GC gc, char *cadena, int offs, char
tecla_ascii, int max );
```

display, window, gc = display, ventana y gc a usar.
Cadena = dónde guardar la cadena que se lea.
Offs = Offset actual dentro de la cadena. Debe ser cero inicialmente y es devuelto por la función en cada paso.
tecla_Ascii = mediante este parámetro se le pasa a la función la tecla pulsada.
Max = número máximo de caracteres a aceptar.

Los keycodes y keysyms no distinguen entre mayúsculas y minúsculas

Internamente, la función realiza lo siguiente:

```
EditaTexto
{
  si offs == 0
     DibujaRecuadro;
     DibujaCursor;
  fin si

  si pulsacion está entre 32 y 127
     cadena[offs] = tecla;
     cadena[offs+1] = '\0';
     BorrarViejoCursorAlaIzquierda;
     DibujarTeclaPulsada;
     DibujarNuevoCursor;
     offs++;
  fin si

  si pulsacion == BORRAR
     BorrarCursorActual;
     BorrarLetraAnterior;
     DibujarNuevoCursor;
  fin si

  si tecla == ENTER
     finalizar edicion
  fin si
  return(offs);
}
```

Como puede verse, el cursor no es más que el carácter '_' dibujado antes de trazar la letra, y la manera de borrar las letras consiste en volver atrás y dibujarlas en color negro.
Mediante estos parámetros, la llamada a la función de edición de texto sería así:

```
switch( evento.type )
{
   case KeyPress:
      key = XKeycodeToKeysym(display,
evento.xkey.keycode, mays);
      if( key == XK_Shift_L || key == XK_Shift_R )
         mays = !mays;
      offset = EditaTexto( display, window, gc,
texto, offset, key, 40 );
      break;

   case KeyRelease:
      key = XKeycodeToKeysym(display,
evento.xkey.keycode, mays);
      if( key == XK_Shift_L || key == XK_Shift_R )
         mays = !mays;
      break;
}
```

Como puede verse, cuando se produce una pulsación de tecla (*KeyPress*) se trata de obtener el *KeySym* que equivale a dicha pulsación (los *KeySym* son nombres genéricos que representan a las teclas independientemente del hardware, siendo genéricas así para todas las versiones de X) y se le pasa a la función de edición para que la incorpore en la cadena. La manera de realizar esta conversión es mediante la función

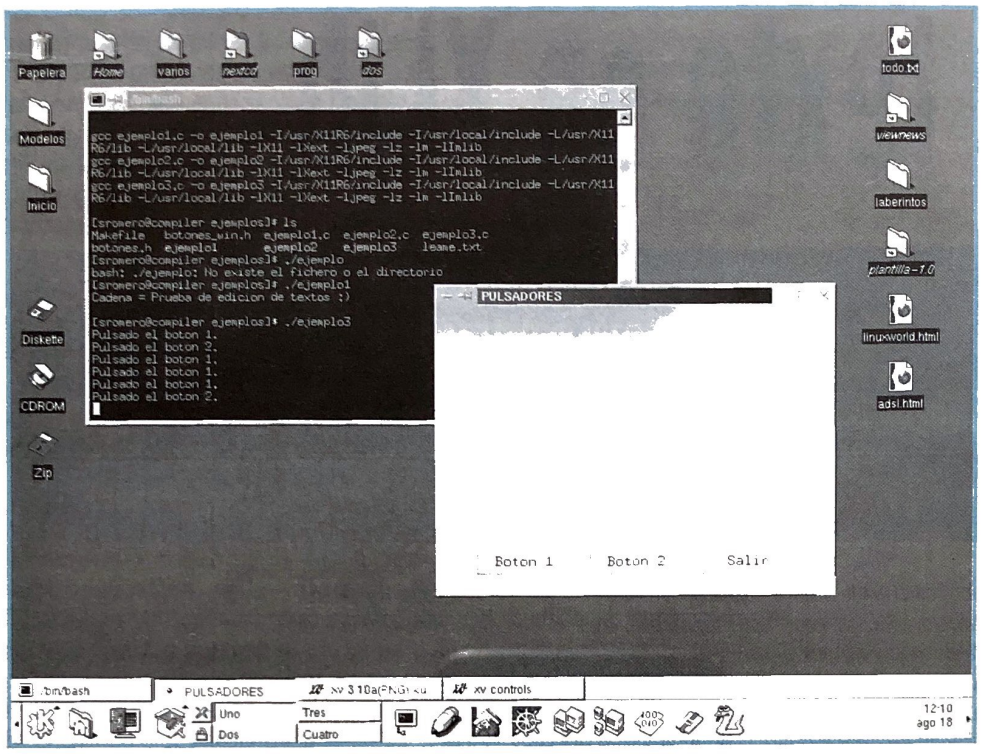


Figura 2. Uso de los pulsadores.

XKeycodeToKeysym() que vemos a continuación:

```
#include "X11/keysym.h"

KeySym XKeycodeToKeysym(display, keycode, index)
Display *display;
KeyCode keycode;
int index;
```

Esta función acepta el *keycode* de una tecla y devuelve el *KeySym* que le corresponde. El parámetro *index* puede ser 0 ó 1, y especifica si deseamos obtener el *KeySym* en mayúsculas o en minúsculas, ya que los *keycodes* no distinguen entre estos 2 casos. Para los *keycodes*, la pulsación de mayúsculas más una tecla es equivalente al código de *SHIFT* seguido del código de la tecla, como 2 teclas independientes, de modo que no se obtiene la tecla en mayúsculas.
Como los *KeySym* no distinguen entre las mayúsculas y las minúsculas, lo que hacemos es utilizar una variable *Mays* que variaremos entre 0 y 1 cuando se detecte la pulsación de cualquiera de los 2 *SHIFTS*, como se puede ver en el *KeyRelease*.
En el ejemplo se realiza un tipo de edición muy sencillo, pero nada nos impide modificar la función para que permita especificar la X e Y de inicio, colores de texto y fondo, hacerla más eficiente, soportar otras teclas o incluso hacerla asíncrona (sin necesidad de depender del bucle de eventos), como vimos en la anterior entrega. Para cualquier programa serio, será necesario modificar esta rutina para que sea más robusta, para que permita reconstruir la cadena completa (en lugar de trabajar letra a letra) y para que tenga un modo de funcionamiento más eficiente, pero la base puede ser similar a la vista. Una porción de código algo especial que puede observarse en el **Listado 1** que se incluye en el CD-Rom de la revista, es el siguiente (antes del bucle de eventos):

```
while (1)
{
  XNextEvent(display, &evento);
  if (evento.type == Expose) break;
}
```

Mediante esta sección de código descartamos todos los eventos ocurridos hasta que se genere el primer *Expose*, el cual se produce cuando se mapea la ventana. Con esto conseguimos que los eventos que nos interesan comiencen a procesarse a partir de la creación y visualización completa de la ventana.

INTERFAZ PARA BOTONES

Unos de los elementos más útiles que podemos encontrar en los *toolkits* de programación, son los pulsadores o botones. Estos elementos son pequeños rectángulos que al pasar sobre ellos con el puntero del ratón y presionar el botón del mismo, realizan una determinada acción (ver **Figura 2**). Mediante el uso de pulsadores es muy fácil organizar los programas para que el usuario realice las tareas de una manera muy intuitiva. Con XLib no disponemos de funciones específicas para la creación y mantenimiento de pulsadores, pero se pueden crear perfectamente a partir de las funciones básicas de que disponemos.
En el **Listado 2**, el cual lo podemos observar en el CD-Rom que acompaña la revista, disponemos de una sencilla librería de botones con un aspecto muy rudimentario pero que cumplen perfectamente su función y están organizados de forma que el lector puede modificar sus funciones de dibujado y crear los botones a su deseo (mediante *pixmaps*, rectángulos rellenos, etc.). Las funciones y variables utilizadas son las siguientes:

Boton botones[BOTON_MAX];

Estructura que almacenará los datos de hasta *BOTON_MAX* botones (*display*, ventana madre, etc.).

int botones_usados = 0;

Número de botones usados hasta el momento.

*Int InicializarBotones(Display *display, Font fuente);*

Los KeySym son nombres genéricos que representan a las teclas en X

Esta función inicializa la librería de botones calculando los valores para determinados colores y la altura del texto que usaremos (la fuente con que inicializamos la librería ha de ser la misma que usaremos para los botones).

*int CrearBoton(Display *display, Window window, int x, int y, Font fuente, char *texto, int (*funcion)());*

Función que crea un nuevo botón y le asigna la función que se debe ejecutar ante su pulsación. Lo único que hace internamente es buscar el siguiente hueco en la estructura *botones*, dentro del cual inserta todos los datos relativos al nuevo botón (creado mediante una ventana de un determinado color de texto y fondo). Entre esos datos destaca la función que se ejecutará cuando el botón sea pulsado. La ventana que crearemos recibirá sólo eventos *Expose* y *ButtonPress*, que son los que nos interesan para el funcionamiento del botón.

*int EventoBoton(Display *display, XEvent *evento);*

Esta función la llamaremos desde nuestro bucle de eventos, cuando se reciba un evento *Expose* o un evento *ButtonPress*. Lo único que hace es recorrer todos los botones disponibles, hasta averiguar si el lugar donde se ha pulsado el boton se corresponde con alguno de los botones (no olvidemos que los botones que hemos creado son simples ventanas, de modo que bastará con comprobar si la ventana sobre la que se pulsó el botón del *mouse* es alguna de las ventanas-botón que hemos creado). En caso afirmativo se ejecutará la función *EjecutarBoton()*. Para los eventos *Expose* ocurre lo mismo, pero llamando a la función *RedibujarBoton*. En caso de que se detecte que la pulsación no corresponde a ninguno de los botones, se vuelve a la función llamadora (la del bucle de eventos) para que sea ésta la que procese el evento. De esta forma, los botones se gestionan sólos sin necesidad de código extra por nuestra parte.

*int EjecutarBoton(Display *display, int cual);*

Esta función simplemente dibuja un sencillo sombreado para marcar que el botón ha sido pulsado, y llama a la función asociada al botón, pasándole como parámetros el *display* y la ventana madre del botón. Dentro de esa función podemos hacer cualquier cosa, desde cambiar valores de variables, hasta escribir en pantalla, etc.

Es posible crear un botón mediante el simple uso de una ventana

De esta forma podemos crear botones para Salir, Guardar/Cargar, cambiar opciones, etc, con un mínimo coste de programación.

*int RedibujarBoton(Display *display, int cual);*

Esta última función de la librería redibuja el botón cuando éste lo necesita, es decir, cuando ha sido borrado por cualquier circunstancia (tras un evento *Expose*) y necesita ser redibujado. Actuando sobre esta función y la anterior es posible cambiar totalmente el aspecto de los botones en cada aplicación que programemos.

CREANDO LOS BOTONES

En el **Listado 2** que acompaña la revista, se crean 3 botones con sus 3 funciones asociadas y se permite al usuario que interactúe con la aplicación. Los 2 primeros botones muestran un mensaje, mientras que el tercero hace finalizar la aplicación. La manera de utilizar la librería mediante un ejemplo sencillo es la siguiente:

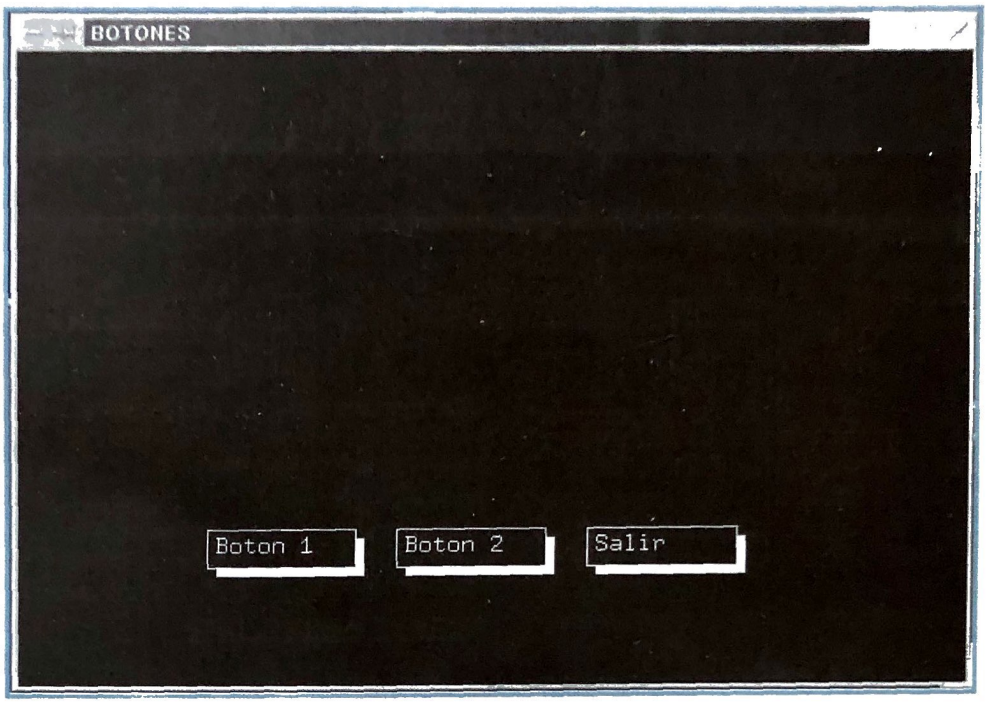


Figura 3. Aspecto de los pulsadores.

Inicialmente declaramos la función de activación para el botón:

```
int FuncionBoton1( Display *, Window );
int FuncionBoton1( Display *display, Window
window )
{
  printf("Pulsado el boton 1.\n");
}

int FuncionBoton2( Display *, Window );
int FuncionBoton2( Display *display, Window
window )
{
  printf("Pulsado el boton 2.\n");
}
```

En cualquier momento durante el inicio del programa, inicializamos nuestra librería de botones (necesitamos haber cargado antes una fuente de texto):

La sección de creación del botón y recogida de eventos quedaría de la siguiente manera:

```
/*despachar todos los eventos antes del 1er expose*/
while (1)
{
   XNextEvent(display, &evento);
   if (evento.type == Expose) break;
}

CrearBoton(display, window, 120, 300, fuente,
"Boton 1", FuncionBoton1 );
CrearBoton(display, window, 240, 300, fuente,
"Boton 2", FuncionBoton2 );

/* bucle de eventos: */
while ( !done )
{
  XNextEvent(display, &evento);

  switch(evento.type)
  {
    case Expose:
      EventoBoton( display, &evento );
      break;

   case ButtonPress:
      EventoBoton( display, &evento );
      break;

   case KeyPress:
      key = XKeycodeToKeysym(display,
evento.xkey.keycode, 0);
      if( evento.xkey.keycode == ESC_KEY )
        done = 1;
        break;
  }
}
```

Como puede verse, simplemente llamamos a *CrearBoton* tantas veces como botones deseemos crear y dentro de la recepción de los eventos *Expose* y *ButtonPress* llamamos a la función que gestiona los eventos. No importa cuántos botones hayamos creado: todos se gestionan mediante las llamadas que se ven arriba y todos se auto-redibujan, borran y ejecutan en el momento en el que el usuario pulsa. Simplemente tendremos que tener en cuenta si *EventoBoton()* devuelve 1 ó 0, para saber si ha gestionado ella el evento

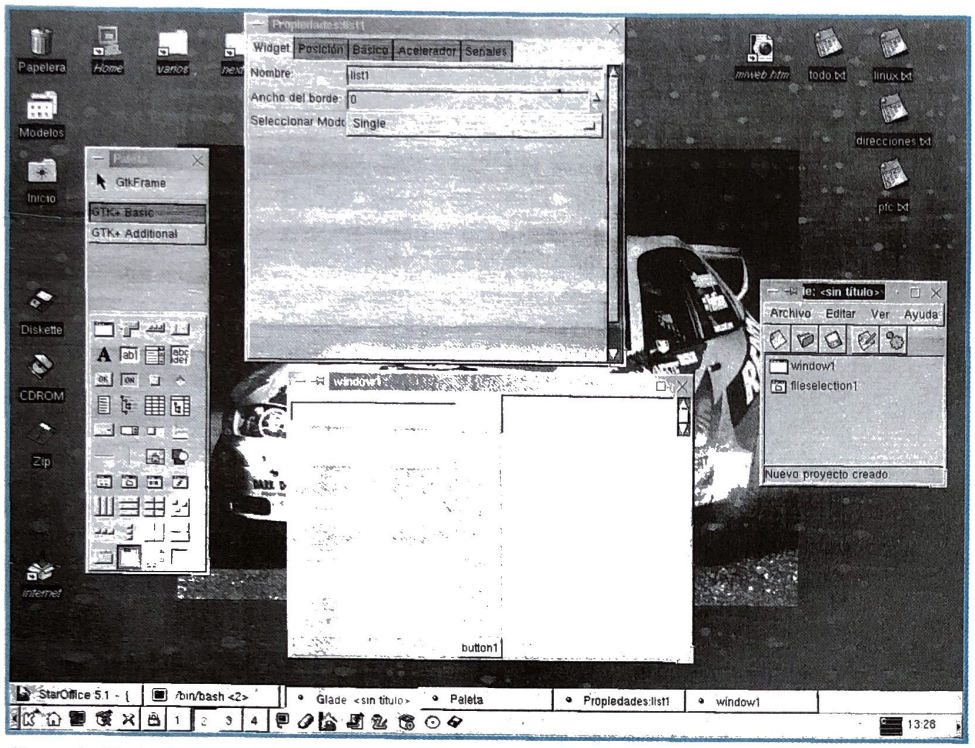


Figura 4. Glade durante el desarrollo de una aplicación.

(era una pulsación o *Expose* del botón) o si no lo era y es un evento para nosotros.
En las funciones de ejecución de los botones puede realizarse cualquier acción (en nuestro caso simplemente hemos hecho una impresión a pantalla), pero en una aplicación normal podría perfectamente guardarse un fichero, hacer un cálculo, lanzar otra aplicación, etc. Los botones dan muchas posibilidades al programador puesto que puede hacer muchas tareas olvidándose de las funciones de detección de pulsaciones, redibujado, etc., de las que ya disponemos.
Este ejemplo (la librería de pulsadores) también debería servir como ejemplo de cómo se pueden crear *toolkits* para XLib.
Además de botones, podríamos crear una librería con funciones para *combo boxes*, cajas de selección, *radio buttons*, barras de desplazamiento, etc., de forma que las pudieramos usar en nuestros programas.
Claro está que nuestro ejemplo es muy sencillo (ni siquiera utiliza memoria dinámica para la creación de los nuevos botones, sino que usa un vector estático de botones), pero nada nos impide realizarnos nuestras propias funciones para XLib o utilizar las ya existentes en otros *toolkits*.
Por otra parte, el diseño que hemos realizado para los pulsadores (sencillos rectángulos de texto+fondo con una pequeña sombra para las esquinas), los hace similares en aspecto a los de Windows (donde el botón es gris, el texto negro, y las líneas superior e izquierda del rectángulo son de color blanco, y la inferior y derecha de color gris oscuro, excepto cuando es pulsado y dichos colores son intercambiados), pero podrían ser modificados para parecerse a los de cualquier otro sistema gráfico (ver **Figura 3**), incluyendo la posibilidad de dibujar botones con *pixmaps* de fondo. Si hubiesemos utilizado un *toolkit*, probablemente estaríamos limitados a lo que el *toolkit* nos ofreciera, aunque la programación fuera infinitamente más sencilla. También existen herramientas de programación visual (o *RAD*, al estilo Delphi, Visual Basic, etc.), como *Glade o KDevelop*, las cuales reducen el tiempo de programación al mínimo, trabajándose bastante en el ámbito del diseño de las aplicaciones como si de un programa de dibujo se tratase. Esta librería que hemos creado hace algo similar a eso, ya que sólo tenemos que preocuparnos de diseñar la función de ejecución del botón que deseamos. También sería posible diseñar (mediante ventanas cuyos eventos se gestionen) las celdillas de edición de textos, y sin duda será un buen ejercicio para el lector el convertir la función de edición de nuestra primera parte del artículo, en algo similar a un *array* de ventanas preparadas exclusivamente para gestionar *Expose*, *KeyPress* y *KeyRelease*.
Además, empresas como Borland están portando sus *Entornos Integrados de Desarrollo* y sus RADs a Linux, como es el caso de Java Builder, Delphi, Borland C Builder, etc. Sus equivalentes libres, *Glade* y *Kdevelop*, no van retrasados, pues con ellos pueden hacerse aplicaciones de gran calidad, ya sea usando las librerías GTK (*glade*) o QT (*Kdevelop*) en tiempos récord y con funcionalidades máximas. Además, los códigos generados por estos programas son adaptables a las versiones Windows de las mismas librerías, lo cual nos da la posibilidad de realizar nuestros programas para ambas plataformas con el mínimo esfuerzo.

EN RESUMEN

Con este artículo llegamos al final de esta serie sobre programación en X-Window que pretendía acercar el mundo de la programación en X a los lectores de **Programación Actual**. X-Window (y Linux) es un magnífico entorno de programación (no hay más que ver que tenemos la ayuda de todas las funciones incluidas en las páginas *man* del sistema), con un soporte muy completo y con gran cantidad de ejemplos y aplicaciones.
Además, siempre podemos descargar de Internet el código fuente de cualquier programa de X-Window y mirarlo para aprender a realizar determinadas acciones.

X-Window es un entorno de programación con un soporte muy completo

Aquellos programadores de Microsoft Windows que hayan seguido el curso probablemente habrán encontrado misteriosamente parecidos todos los nombres, sistemas de eventos, funciones y estructuras que posteriormente "creó" Microsoft para el desarrollo de su Sistema Operativo, muy parecido internamente a X aunque no haya adoptado de él su estabilidad y funcionamiento puramente de red.
Probablemente ahora muchos lectores se den cuenta del gran potencial de X-Window (sobre todo ahora que empieza a rodar la versión XFree86 - 4), y muy pronto el sector de programación de aplicaciones X se abrirá en el mundo, ya que cada vez más, la gente instala Linux en sus casas cuando antes sólo tenía lugar en los sistemas servidores de Internet.
Y como todo campo que se abre, puede dar muchos beneficios a quienes sepan aprovecharlo. Empresas como Corel, ID Software o StarDivision ya se han dado cuenta de ello, y los programadores *freelance* también. Espero que este curso haya servido para orientar al lector de tal modo que pueda ya escribir aplicaciones X, sabiendo que tendrá el soporte de toda una comunidad con estilo de vida GNU.