Introducción a la programación en X Window y Motif

Modesto F. Castrillón Santana
J.Daniel Hernández Sosa
J.Javier Lorenzo Navarro

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Introducción a la Programación en X Window y Motif

Modesto F. Castrillón Santana
José Daniel Hernández Sosa
José Javier Lorenzo Navarro

ISBN 84-8498-832-5
Depósito Legal GC 1398-1997

Impreso en Las Palmas de Gran Canaria a 30 de Septiembre de 1997

Departamento de Informática y Sistemas
Universidad de Las Palmas de Gran Canaria
España

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

DEC, Hewlett Packard, Sun, AT&T, IBM, Macintosh, Unix, X Window y Motif son marcas registradas.

Índice General

1 Introducción a X Window **1**

1.1 Introducción 1

1.1.1 Origen 1

1.1.2 Ventajas e inconvenientes 1

1.1.3 Requerimientos hardware 2

1.1.4 Otros sistemas de ventanas 2

1.2 Arquitectura X 3

1.2.1 Modelo cliente-servidor 3

1.2.2 Entorno de trabajo 4

1.2.3 Configuraciones especiales 5

1.3 Administración de X Window 5

1.3.1 Arranque y parada 7

1.3.2 Seguridad 11

1.4 Administración de los recursos 11

1.4.1 Opciones en línea de comandos 13

1.4.2 Uso del cliente `xrdb` 13

1.4.3 Recursos especiales 14

1.4.4 Fuentes en X 15

1.5 El manejador de ventanas 15

1.5.1 Configuración general 16

1.5.2 Menús desplegables, funciones y módulos 18

1.6 Clientes especiales y utilidades X 21

1.7 Introducción a la programación en X 22

1.7.1 Niveles de software 23

1.7.2 Protocolo X 23

1.7.3 Programa básico 24

2 Introducción a la programación en X Window **25**

2.1 Introducción 25

2.2 Un primer programa 25

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

2.3 Conceptos y filosofía 28
2.4 Las ventanas 29
2.4.1 Creación de ventanas 29
2.4.2 Visualización de ventanas 31
2.4.3 Destrucción de ventanas 32
2.4.4 Aplicaciones con varias ventanas 32
2.4.5 Geometría de una ventana 33
2.4.6 Características propias de una ventana 33
2.4.7 Atributos de una ventana 34
2.4.8 Configurando ventanas 37
2.4.9 Modificando atributos 37
2.4.10 Eventos y ventanas 37
2.5 Gráficos, GC y Texto 38
2.5.1 Dibujos 38
2.5.2 El Contexto Gráfico 40
2.5.3 Las fuentes y el texto 43
2.5.4 Regiones 46
2.5.5 Imágenes 46
2.5.6 Cursores 47
2.6 Color 48
2.7 Eventos 54
2.8 Teclado y Puntero 58
2.8.1 Teclado 58
2.8.2 El puntero 59
2.9 Manejando recursos 60

3 Programación en Motif 61
3.1 Introducción 61
3.2 Configuración de los widgets 64
3.2.1 Configuración por parte del usuario 65
3.2.2 Configuración por Programa 65
3.3 Cadenas de Caracteres y Fuentes 66
3.4 Callbacks 68
3.5 Jerarquía de los widgets 70
3.6 Widgets Shell 71
3.7 Widgets Contenedores 72
3.7.1 Tablón 72
3.7.2 Formulario 73

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

3.7.3 Apilados . . . 75
3.7.4 Fila Columna . . . 76
3.7.5 Ventana Deslizante . . . 77
3.7.6 Ventana Principal . . . 79
3.7.7 Marco . . . 80
3.7.8 Contenedor . . . 81
3.7.9 Block de Notas . . . 83
3.8 Controles . . . 86
3.8.1 Etiquetas y Separadores . . . 86
3.8.2 Botones . . . 87
3.8.3 Listas . . . 91
3.8.4 Escalas . . . 92
3.8.5 Textos . . . 93
3.9 Menú y Opciones . . . 96
3.9.1 Menús . . . 96
3.9.2 Selección Combinada . . . 101
3.9.3 Selección Rotatoria . . . 101
3.10 Cuadros de diálogo . . . 104
3.10.1 ErrorDialog, InformationDialog, WarningDialog, WorkingDialog y QuestionDialog . . . 104
3.10.2 FileSelectionDialog . . . 105
3.10.3 PromptDialog . . . 106
3.10.4 SelectionDialog . . . 107

Apéndice **111**

aloja_color.c . . . 111
apilado.c . . . 113
backing_store.c . . . 114
block.c . . . 115
boton.c . . . 117
botones.c . . . 117
colormap.c . . . 120
combo.c . . . 122
contenedor.c . . . 123
cursor.c . . . 125
cursor_propio.c . . . 126
dialogos.c . . . 128
escala.c . . . 131
etiqueta.c . . . 132

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

fila-columna.c . . . 133
formulario.c . . . 133
fuente.c . . . 135
hola-motif.c . . . 136
hola-xlib.c . . . 136
marco.c . . . 137
marco.c . . . 139
menus.c . . . 140
mueve.c . . . 144
mueve_hija.c . . . 145
mueve_recta.c . . . 147
pido_color.c . . . 148
pixmap-bitmap.c . . . 150
pixmap-fondo.c . . . 151
pixmap-fondo-None.c . . . 152
profundidad-visual.c . . . 153
propagate.c . . . 155
rectas.c . . . 156
spin.c . . . 157
tamaño.c . . . 159
teclas.c . . . 161
vent-deslizante.c . . . 162
ventana-pral.c . . . 163
ventana-hermanas.c . . . 165
ventanas-padre-hija.c . . . 166

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Índice de Figuras

2.1 Una ventana 28
2.2 Una comunidad de ventanas 29
2.3 Dos ventanas hermanas a la izquierda, y padre e hija a la derecha 33
2.4 Un bitmap para el fondo 35
2.5 Otra fuente de texto 44

3.1 Progama "Hola Mundo" 62
3.2 Estructura de una XmString 67
3.3 Progama "Boton" 69
3.4 Nivel más alto del jerarquía de widgets 71
3.5 Jerarquía de widgets Shell 71
3.6 Programa basado en Bulletin Board 73
3.7 XmForm con cuatro botones 74
3.8 XmForm con cuatro botones despues de cambiar tamaño 75
3.9 Widget Paned con dos botones 76
3.10 Widget RowColumn 77
3.11 Widget RowColumn con orientación horizontal 77
3.12 Widget Scrolled Window conteniendo 6 etiquetas 78
3.13 Ventana principal de las aplicaciones en Motif 79
3.14 Widget XmForm con un marco 81
3.15 Widget Container 82
3.16 Widget Container con todos las entradas desplegadas 83
3.17 Aplicación basada en el widget Notebook 84
3.18 Widgets de tipo XmPrimitive 86
3.19 Widgets Labels con distinta justificación 87
3.20 Ventana con diferentes tipos de botones 88
3.21 Widget Scale 92
3.22 Widgets de texto en Motif 94
3.23 Distintos tipos de menu 98
3.24 Menu Popup 100

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

3.25 Salida del programa `combo.c` 102
3.26 Widget SpinBox 102
3.27 ErrorDialog 105
3.28 InformationDialog 105
3.29 WarningDialog 106
3.30 WorkingDialog 106
3.31 QuestionDialog 107
3.32 FileSelectionDialog 108
3.33 PromptDialog 109
3.34 SelectionDialog 109

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Capítulo 1

Introducción a X Window

1.1 Introducción

El *sistema de ventanas X*, también conocido como *sistema X*, *X Window* o simplemente *X*, es un sistema gráfico de ventanas concebido para operar en red y que ha sido adoptado como estándar en la industria. El diseño de X permite independizar la programación de las aplicaciones de los detalles relativos al hardware y el sistema operativo sobre el que se van a ejecutar. De hecho existen implementaciones de X para prácticamente cualquier tipo de plataforma hardware, desde PC o Macintosh hasta supercomputadores. Esta expansión se ha visto favorecida por el hecho de que X es de libre distribución.

X está soportado actualmente por un importante consorcio del que forman parte fabricantes como DEC, Hewlett-Packard, Sun, IBM o AT&T.

La información recogida en estas notas corresponde a la implementación de X Window*XFree86* para Linux. En otras implementaciones, si bien se mantienen los conceptos generales, pueden variar aspectos concretos como nombre y ubicación de ficheros de configuración, mecanismos de inicialización, etc.

1.1.1 Origen

X se comienza a gestar en 1984 a partir de un proyecto del MIT, el Proyecto Athena, que pretendía la interconexión de estaciones de trabajo gráficas de diferente naturaleza y sistema operativo. En dicho proyecto participaban diferentes empresas como IBM o DEC.

La primera versión del sistema X se lanza de forma gratuita en 1988, bajo la denominación *X11 R2* (versión 11, segunda edición). Tras este primer producto surgieron diferentes actualizaciones hasta llegar a la más reciente, la sexta edición de 1995.

1.1.2 Ventajas e inconvenientes

Siguiendo la filosofía general de los sistemas de ventanas, X divide la pantalla en regiones o ventanas independientes sobre las que se pueden realizar operaciones de entrada y salida.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

La interacción del usuario con las aplicaciones se vuelve más intuitiva, simplificándose el diseño de las interfaces. La popularidad del entorno reporta además beneficios a largo plazo como son la gran cantidad de software disponible o la reducción del tiempo de aprendizaje en el manejo de nuevas aplicaciones, derivada del aspecto uniforme que éstas poseen.

Uno de los objetivos principales de X es la transportabilidad de las aplicaciones entre las diferentes plataformas para las que se han desarrollado implementaciones de este estándar. Una aplicación que haga uso de los recursos básicos de X puede migrar a otro equipo y funcionar de forma correcta, una vez compilada, lógicamente, sin requerir modificación alguna en su código.

La naturaleza orientada a red de X permite que no existan limitaciones en cuanto a la ejecución de una aplicación en una máquina determinada y su visualización en otra diferente. Esto contribuye a una mejor y más cómoda distribución de recursos, mejorando la productividad del sistema. X posee mecanismos que evitan la sobrecarga que en principio podría suponer el intercambio de información gráfica entre varias máquinas.

Existen también algunos inconvenientes en el uso de X que derivan fundamentalmente de la complejidad que resulta de intentar cubrir objetivos como son la independencia del hardware o la naturaleza distribuida. Esta complejidad se traduce en elevados consumos de recursos del tipo espacio en disco, memoria, cpu, ancho de banda de red, etc.; demandas que precisan equipos con prestaciones elevadas si se desea conseguir un entorno de trabajo aceptable.

1.1.3 Requerimientos hardware

La implementación de X para ordenadores personales utilizada en este trabajo es la denominada XFree86, y está disponible para System V/386, 386BSD, Linux, y otros sistemas. En el caso de Linux, los requerimientos mínimos corresponden a un PC 486 con 8 Mb de RAM y una tarjeta de vídeo soportada por el servidor X. Para una operación más cómoda se recomienda incrementar la cantidad de memoria RAM instalada así como disponer de una tarjeta con acelerador gráfico.

1.1.4 Otros sistemas de ventanas

Además de X Window, otras interfaces gráficas de usuario persiguen objetivos comunes como son el uso de elementos de control visuales, la manipulación directa de objetos sobre la pantalla, la homogeneidad en la apariencia y operación de las aplicaciones, etc. Algunas de las más importantes son Macintosh Finder, Microsoft Windows, OS/2 Presentation Manager, DESQview o NeXTstep.

En comparación con estos sistemas, X Window es un sistema más versátil en lo que se refiere al aspecto final de las aplicaciones generadas. Otra característica diferenciadora es que el sistema X ha sido diseñado para operar en red interconectando máquinas de elevadas prestaciones.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

1.2 Arquitectura X

Como ya se ha indicado X es un sistema concebido para operar a través de una red de interconexión de máquinas heterogéneas. En este entorno, uno de los principales factores de diseño a considerar es la minimización de las comunicaciones dentro de lo posible, a fin de no degradar en exceso los tiempos de respuesta de las aplicaciones. Como veremos, la solución adoptada en X se basa en la distribución de las diferentes tareas asociadas a la ejecución de una aplicación entre distintos procesos.

1.2.1 Modelo cliente-servidor

La arquitectura X se basa en el *modelo cliente-servidor*. El sistema está integrado por dos componentes básicos, los *clientes* o *aplicaciones X* y los *servidores* o *display servers*, que se reparten las tareas asociadas con la ejecución de las aplicaciones en el entorno. Dichos elementos se comunican a través de la red intercambiando mensajes de petición o *requests*, que van desde los clientes a los servidores, y de respuesta o *replies*, que se dirigen desde los servidores a los clientes. Los mensajes obedecen a un protocolo de comunicación definido, el *protocolo X*.

El servidor es el proceso que actúa de intermediario entre los clientes y los dispositivos hardware y el sistema operativo de la máquina, ocultando los detalles específicos de cada entorno. Se encarga de llevar a cabo diversas tareas como son:

- Gestionar el acceso a los dispositivos de visualización cuando existen varios clientes ejecutándose.
- Atender los mensajes de petición que le llegan y responder a los mismos.
- Procesar la información de entrada que se genera (acciones del ratón, pulsaciones del teclado, etc.) enviándola al cliente destinatario a través de mensajes de eventos.
- Mantener estructuras de datos actualizadas con información relativa a los diferentes recursos que están siendo utilizados (ventanas, fuentes, iconos, etc).

Los clientes son aplicaciones que llevan a cabo tareas específicas interaccionando con el usuario de forma indirecta. La información de entrada, como son las acciones ejecutadas por el usuario sobre la interfaz, es canalizada hacia la aplicación a través del servidor X. Análogamente, la información de salida, como respuestas de la interfaz o resultados de la aplicación, es representada sobre la ventana o ventanas de la aplicación a través del servidor X, que acepta primitivas gráficas de dibujo.

Algunos clientes X cumplen cometidos especiales dentro del entorno. El más relevante es el *window manager* (manejador de ventanas), encargado de dotar al sistema X de una determinada apariencia y funcionalidad. Este cliente se tratará en detalle en la sección 1.5.

La distribución de tareas y de datos entre el cliente y el servidor aporta diversos aspectos positivos como son el mejorar la reactividad del sistema, reducir su sobrecarga, disminuir la cantidad de datos a transmitir por la red, etc.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

El modelo cliente-servidor utilizado en el sistema X difiere ligeramente del esquema tradicional empleado en otras arquitecturas. Por un lado, el usuario no interactúa directamente con la aplicación cliente, sino que lo hace a través del servidor que opera como intermediario. Por otra parte el usuario se ubica en la máquina donde se ejecuta el servidor, mientras que las aplicaciones clientes corren sobre máquinas remotas.

1.2.2 Entorno de trabajo

Un entorno de trabajo para el sistema X está constituido básicamente por un teclado, elementos de visualización y apuntadores. La visualización se realiza a través una o varias pantallas con capacidad gráfica, tanto en color como monocromas. Los dispositivos apuntadores facilitan la navegación por el entorno multiventana, existiendo una amplia variedad que incluye ratones, track-balls, tabletas digitalizadoras, lápices ópticos, space-balls, etc. Con frecuencia se utilizará el término *display* como equivalente a servidor X para hacer referencia a un conjunto de monitor/es, teclado y ratón conectados a una misma máquina.

La asociación entre un cliente y un servidor X se establece a través de la variable DISPLAY que posee el siguiente formato

```
nombre_máquina:número_display.número_pantalla
```

donde `nombre_máquina` hace referencia a la máquina a la que está conectado físicamente el display, `número_display` identifica un display en concreto (generalmente tomará el valor 0), y `número_pantalla` identifica una pantalla en concreto (también valdrá 0 en la mayoría de las ocasiones). Si se supera el control de acceso la aplicación se visualizará sobre el servidor indicado por la variable DISPLAY en el momento de su ejecución.

El tipo de canal de comunicación que se establece entre el cliente o aplicación X y el servidor viene reflejado en la parte reservada al nombre de la máquina dentro de la variable DISPLAY. Si no se especifica ningún nombre el display es local, y el sistema selecciona la vía de comunicación más eficiente de las disponibles. Si el nombre corresponde a una dirección IP, la comunicación se establece a través del protocolo TCP/IP. Por último, si el nombre va seguido por "::", el canal utilizado será DECnet.

El teclado se caracteriza en X mediante la distinción de dos niveles de codificación. El primero corresponde al conjunto denominado *keycodes*, que son específicos de cada servidor y contiene información relativa a las teclas físicas de que está compuesto el teclado. El segundo nivel está asociado al conjunto denominado *keysyms*, que es independiente del servidor y hace referencia a los distintos símbolos que pueden aparecer sobre el teclado. La correspondencia entre teclas y símbolos se obtiene con ayuda de dos tablas denominadas tabla de modificadores, para teclas de control como `shift`, `ctrl`, etc.; y tabla de correspondencia, que permite traducir el código de la pulsación en un símbolo determinado.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

1.2.3 Configuraciones especiales

Una configuración alternativa es la de las terminales X. Se trata de terminales gráficos sin disco sobre los que corre únicamente un proceso que es el servidor X. Esto permite aumentar el número de puestos de trabajo a un coste reducido, aunque el rendimiento que se obtiene se degrada tanto por el aumento de las comunicaciones remotas a través de la red como por la sobrecarga de la estación de trabajo servidora.

Una segunda posibilidad, que evita tener que realizar nuevas inversiones en hardware, es la de las terminales X emuladas por software sobre equipos basados en otros sistemas de ventanas. En estos casos el rendimiento también puede llegar a reducirse sensiblemente.

1.3 Administración de X Window

La administración del sistema X hace referencia a aspectos fundamentales como son la instalación, configuración hardware, configuración software o mantenimiento. Aquí nos centraremos especialmente en la configuración software del entorno y los clientes a través de ficheros de configuración y recursos. Los recursos se volverán a analizar en detalle en la sección 1.4.

La instalación del sistema X Window es muy dependiente de cada tipo de distribución, aunque no suele plantear problemas. El siguiente paso es la configuración en función del hardware, esto es, la combinación tarjeta gráfica-monitor. En nuestro caso, Xfree86 para Linux, la configuración se fija en el fichero `XF86Config` que consta de diferentes secciones para el teclado, ratón, monitor, tarjeta gráfica, etc. Este fichero se puede ubicar en diferentes directorios como `/etc` o `/usr/lib/X11`.

Existen diferentes utilidades como `SuperProbe` o `xf86config` que pueden ayudar a la generación de este fichero. También se dispone de diferentes ficheros con parámetros ejemplo para diferentes configuraciones, como `Monitors` o `VideoModes.doc` en el directorio `/usr/lib/X11/doc`.

Ejemplo: Fragmentos de un fichero de configuración hardware típico.

```
# **********************************************************************
# Files section.  This allows default font and rgb paths to be set
# **********************************************************************

Section "Files"
    RgbPath   "/usr/X11R6/lib/X11/rgb"
    FontPath  "/usr/X11R6/lib/X11/fonts/misc/"
    FontPath  "/usr/X11R6/lib/X11/fonts/Type1/"
    FontPath  "/usr/X11R6/lib/X11/fonts/Speedo/"
    FontPath  "/usr/X11R6/lib/X11/fonts/75dpi/"
    FontPath  "/usr/X11R6/lib/X11/fonts/100dpi/"
EndSection

# **********************************************************************
# Keyboard section
# **********************************************************************

Section "Keyboard"
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
    Protocol    "Standard"
    AutoRepeat 500 5
    RightAlt    ModeShift
EndSection

# **********************************************************************
# Pointer section
# **********************************************************************

Section "Pointer"
    Protocol    "MouseSystems"
    Device      "/dev/mouse"
EndSection

# **********************************************************************
# Monitor section
# **********************************************************************

Section "Monitor"
    Identifier  "nec"
    VendorName  "nec"
    ModelName   "nec"
    HorizSync   27-57
    VertRefresh  55-90
    Modeline "640x400"     25.175 640  664  760  800   400  409  411  450
    Modeline "640x480"     25.175 640  664  760  800   480  491  493  525
    ModeLine "800x600"     36     800  824  896 1024   600  601  603  625
    Modeline "1024x768"    44.9   1024 1048 1208 1264   768  776  784  817
 Interlace
EndSection

# **********************************************************************
# Graphics device section
# **********************************************************************

Section "Device"
    Identifier  "diamond"
    VendorName  "diamond"
    BoardName   "diamond"
    Chipset     "mmio-928"
    VideoRam    2048
    Option      "diamond"
    Ramdac      "ti3026"
    Clockchip   "ti3026"
EndSection

# **********************************************************************
# Screen sections
# **********************************************************************

Section "Screen"
    Driver  "svga"
    Device  "diamond"
    Monitor "nec"
    Subsection "Display"
        Depth       8
        Modes       "640x480" "800x600" "1024x768"
        ViewPort    0 0
        Virtual     1024x768
    EndSubsection
EndSection
```

Dentro de `XF86Config` se configuran diferentes resoluciones de trabajo para la pantalla gráfica. Una vez en marcha el sistema se podrá conmutar de unas a otras mediante las combinaciones de teclas `Ctrl+Alt++` y `Ctrl+Alt+-`. También es posible seleccionar otras terminales virtuales pulsando `Ctrl+Alt+F1...F12`.

Dentro ya de la configuración software, existen ficheros de configuración globales para el sistema, que afectan a todos los usuarios en forma de opciones por defecto. Los ficheros del sistema están ubicados normalmente en el directorio `/usr/lib/X11` o en algu-

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

no de sus subdirectorios, por lo que se hará referencia al mismo con frecuencia como "el directorio del sistema". Opcionalmente pueden existir réplicas de dichos ficheros con un nombre similar en los directorios de cada usuario, de forma que las opciones que contengan tienen prioridad sobre las opciones por defecto.

Los mensajes de error del sistema están catalogados en el fichero del sistema `XErrorDB`, y puede activarse o desactivarse su visualización como un recurso más del sistema.

1.3.1 Arranque y parada

Una vez completada la instalación y la configuración hardware, el arranque del sistema de ventanas X implica normalmente la realización de tres pasos que son: arranque del servidor, lanzamiento de al menos un emulador de terminal y arranque del manejador de ventanas. Todas estas operaciones pueden efectuarse bien de forma manual o automática.

Para el arranque manual el usuario ejecuta un guión o *script* especial que suele designarse por diferentes nombres como `startx`, `x11start`, `x11`, etc. Este programa a su vez llama a `xinit`, que es quien realmente se encarga de la inicialización a partir de la información contenida en los ficheros del sistema `xinitrc` y `xserverrc` (o bien `.xinitrc` y `.xserverrc` si existen en el directorio de usuario). El fichero `xserverrc` indica qué servidor debe ejecutarse y con qué opciones, mientras que `xinitrc` hace referencia a los clientes que se van a lanzar inicialmente y a los valores que deben fijarse por defecto para diferentes recursos.

Ejemplo: Fragmentos de algunos de los ficheros que intervienen en el proceso de arranque manual.

*** startx.**

```
userclientrc=$HOME/.xinitrc
userserverrc=$HOME/.xserverrc
sysclientrc=/usr/X11R6/lib/X11/xinit/xinitrc
sysserverrc=/usr/X11R6/lib/X11/xinit/xserverrc
clientargs=""
serverargs=""

if [ -f $userclientrc ]; then
    clientargs=$userclientrc
else if [ -f $sysclientrc ]; then
    clientargs=$sysclientrc
fi
fi

if [ -f $userserverrc ]; then
    serverargs=$userserverrc
else if [ -f $sysserverrc ]; then
    serverargs=$sysserverrc
fi
fi

xinit $clientargs -- $serverargs
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

*** xinitrc.**

```
userresources=$HOME/.Xresources
usermodmap=$HOME/.Xmodmap
sysresources=/usr/X11R6/lib/X11/xinit/.Xresources
sysmodmap=/usr/X11R6/lib/X11/xinit/.Xmodmap

if [ -f $sysresources ]; then
    xrdb -merge $sysresources
fi

if [ -f $sysmodmap ]; then
    xmodmap $sysmodmap
fi

if [ -f $userresources ]; then
    xrdb -merge $userresources
fi

if [ -f $usermodmap ]; then
    xmodmap $usermodmap
fi

xsetroot -solid SteelBlue
fvwm
```

La ejecución del servidor está asociada con el último cliente, que se lanza en primer plano, y que normalmente corresponderá con `xterm` o el manejador de ventanas, de forma que cuando este cliente termina se detiene el servidor.

El arranque automático se configura en el fichero `inittab` sobre un nivel de ejecución determinado para que lo realice el proceso `init` durante la puesta en marcha del sistema, normalmente por medio de un guión `rc`. Dicho guión invoca al programa `xdm` (*X Display Manager*) encargado de la inicialización del sistema X, el cual lee la información contenida en diversos ficheros de configuración de comandos especificados en el fichero `xdm-config`. `xdm` se ejecuta en un bucle infinito, lo que permite que el servidor X esté corriendo continuamente en el sistema, con independencia de que existan o no sesiones X abiertas.

Ejemplo: Fragmentos de algunos de los ficheros que intervienen en el arranque automático.

*** inittab.**

```
id:4:initdefault:

si:S:sysinit:/etc/rc.d/rc.S
...
c3:45:respawn:/sbin/agetty 38400 tty3 linux
c4:45:respawn:/sbin/agetty 38400 tty4 linux
...
x1:4:wait:/etc/rc.d/rc.4
...
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

*** rc.**

```
echo "Starting up the X Window System V.11 R.6..."
exec /usr/X11R6/bin/xdm -nodaemon
```

*** xdm-config.**

```
DisplayManager.errorLogFile: /usr/X11R6/lib/X11/xdm/xdm-errors
DisplayManager.pidFile: /usr/X11R6/lib/X11/xdm/xdm-pid
DisplayManager.keyFile: /usr/X11R6/lib/X11/xdm/xdm-keys
DisplayManager.servers: /usr/X11R6/lib/X11/xdm/Xservers
DisplayManager.accessFile: /usr/X11R6/lib/X11/xdm/Xaccess
DisplayManager.-0.authorize: true
DisplayManager.-0.setup: /usr/X11R6/lib/X11/xdm/Xsetup-0
DisplayManager.-0.startup: /usr/X11R6/lib/X11/xdm/GiveConsole
DisplayManager.-0.reset: /usr/X11R6/lib/X11/xdm/TakeConsole
DisplayManager*resources: /usr/X11R6/lib/X11/xdm/Xresources
DisplayManager*session: /usr/X11R6/lib/X11/xdm/Xsession
DisplayManager*authComplain: false
```

Los ficheros de configuración de `xdm` están asociados con etapas como la inicialización del servidor (`setup`), la carga de los recursos (`resources`), la apertura de una sesión (`startup`), su ejecución (`session`) y finalización (`reset`). El propio `xdm-config` permite configurar diversas opciones como ficheros de volcado de errores o parámetros de seguridad.

Ejemplo: Fichero de recursos para `xdm` `Xresources`.

```
xlogin*login.translations: #override\
  Ctrl<Key>R:       abort-display()\n\
  <Key>F1:          set-session-argument(failsafe) finish-field()\n\
  Ctrl<Key>Return: set-session-argument(failsafe) finish-field()\n\
  <Key>Return: set-session-argument() finish-field()

xlogin*borderWidth: 3
xlogin*greeting:    CLIENTHOST
xlogin*namePrompt:  login:\
xlogin*fail:        Login failed
#ifdef COLOR
  xlogin*greetColor: CadetBlue
  xlogin*failColor:  red
  *Foreground:       black
  *Background:       #fffff0
#else
  xlogin*Foreground: black
  xlogin*Background: white
#endif

XConsole.text.geometry: 480x130
XConsole.verbose:       true
XConsole*iconic:        true
XConsole*font:          fixed

Chooser*geometry:           700x500+300+200
Chooser*allowShellResize:   false
Chooser*viewport.forceBars: true
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
Chooser*label.font:          *-new century schoolbook-bold-i-normal-*-240-*
Chooser*label.label:         XDMCP Host Menu  from CLIENTHOST
Chooser*list.font:           -*-*-medium-r-normal-*-*-230-*-*-c-*-iso8859-1
Chooser*Command.font:        *-new century schoolbook-bold-r-normal-*-180-*
```

El guión que se encarga de la ejecución de la sesión, que suele denominarse `Xsession`, comprueba si existe un fichero de control de la sesión en el directorio de usuario (`.xsession`) para cederle el control. De no ser así se ejecutan una serie de acciones por defecto contenidas en el correspondiente fichero del sistema.

El fichero de control de la sesión realiza diferentes operaciones como son la carga de los recursos por defecto en el servidor, establecimiento de parámetros de configuración, o lanzamiento de clientes iniciales. Cada usuario puede utilizar este fichero para personalizar su entorno de trabajo.

Ejemplo: Ficheros de sesión.

*** Del sistema (`Xsession`).**

```
...
case $# in
  1)
    case $1 in
      failsafe)
        exec xterm -geometry 80x24-0-0
;;
    esac
esac
...
startup=$HOME/.xsession
sysstartup1=/usr/X11R6/lib/X11/xinit/xinitrc
resources=$HOME/.Xresources
if [ -f "$startup" ]; then
  exec "$startup"
elif [ -x "$sysstartup1" ]; then
  exec $sysstartup1
else
  if [ -f "$resources" ]; then
    xrdb -load "$resources"
  fi
  twm &
  exec xterm -geometry 80x24+10+10 -ls
fi
```

*** Del usuario (`.xsession`).**

```
userresources=$HOME/.Xresources
usermodmap=$HOME/.Xmodmap
sysresources=/usr/X11R6/lib/X11/xinit/.Xresources
sysmodmap=/usr/X11R6/lib/X11/xinit/.Xmodmap
if [ -f $sysresources ]; then
  xrdb -merge $sysresources
fi
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
if [ -f $sysmodmap ]; then
  xmodmap $sysmodmap
fi

if [ -f $userresources ]; then
  xrdb -merge $userresources
fi

if [ -f $usermodmap ]; then
  xmodmap $usermodmap
fi

xset s on
xset s 240
xsetroot -solid SteelBlue
fvwm
```

1.3.2 Seguridad

El acceso desde los clientes remotos a un determinado servidor puede gestionarse a través de diversos mecanismos. El esquema más simple es el basado en las listas de control de acceso, que contienen los nombres de los equipos remotos que tienen permitido el acceso al servidor local. Dicha lista se gestiona a través del cliente `xhost`, que permite consultar el estado actual invocándolo sin argumentos (`xhost`), añadir un equipo a la lista (`xhost nombre-equipo`) o eliminarlo (`xhost - nombre_equipo`).

Otros esquemas más elaborados se basan en establecer controles individuales por usuario mediante claves. Algunos ejemplos son MIT-MAGIC-COOKIE-1 (que emplea claves sin codificar), XDM-AUTORIZATION-1 (con encriptación mediante el algoritmo DES), MIT-KERBEROS-5 (con doble autentificación), etc. Sin embargo, hay que tener en cuenta que la utilización del cliente `xhost` para añadir un equipo a la lista de control de acceso tiene prioridad sobre el esquema de seguridad general implantado, que ya no se aplicará para las conexiones procedentes de dicho equipo.

La especificación del tipo de control de acceso a utilizar se realiza en los ficheros de configuración de `xdm` `xdm-config` y `Xresources`. La información relativa a la seguridad se almacena (salvo para el caso de las listas de control de acceso) en el fichero de usuario `.Xauthorize`, y puede ser consultada y modificada con ayuda del cliente `xauth`.

1.4 Administración de los recursos

X proporciona un mecanismo de almacenamiento de valores por defecto para los diferentes recursos que utilizan las aplicaciones. Por recurso se entiende el conjunto de elementos del sistema de los que pueden disponer los diferentes clientes de forma compartida, como son tipos de letra, colores, botones, barras deslizantes, etc. Estos valores se organizan en una base de datos en el servidor que puede ser accedida desde los diferentes clientes, lo que resulta adecuado para un entorno distribuido como el que nos ocupa.

Los clientes obtienen los recursos a través de diversos mecanismos. A continuación se muestra una lista conteniendo las diferentes alternativas ordenadas de mayor a menor prioridad.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

1. Opciones en línea de comandos: sólo se mantienen en esa ejecución de la aplicación.
2. Fichero indicado por la variable de entorno del sistema XENVIRONMENT.
3. Fichero de usuario `$HOME/.Xdefaults-host`.
4. Propiedades de la ventana raíz RESOURCE-MANAGER o SCREEN-RESOURCES.
5. Fichero de usuario `$HOME/.Xdefaults` o fichero del sistema `sys.Xdefaults`.
6. Ficheros contenidos en el directorio especificado por la variable de entorno del usuario XUSERFILESEARCHPATH.
7. Ficheros contenidos en el directorio indicado por XAPPLRESDIR.
8. Información relativa a cada aplicación en el directorio del sistema `app-defaults`.

Los recursos pueden ser modificados a través de diferentes mecanismos como son la carga en la propiedad RESOURCE-MANAGER mediante el cliente X `xrdb`, la edición directa de los diferentes ficheros de configuración, o las opciones en línea de comando.

La sintaxis de las especificaciones de recursos es la siguiente:

```
[nombre_cliente | clase_cliente]*[parte_cliente]*recurso:  valor
```

donde los dos primeros campos determinan qué clientes (o partes de clientes) se ven afectados, el tercer campo designa los recursos a establecer, y el último el valor que se va a asignar.

Ejemplo: Fichero de recursos de aplicación `.Xdefaults`.

```
*Form.background:                   #c0c0c0
*TransientShell*Dialog.background: #c0c0c0
*Command.background:                #c0c0c0
!*Menubutton.background:            #c0c0c0

*ScrollbarBackground:               #c0c0c0
*Scrollbar*background:              #c0c0c0
*Scrollbar*width:                   15
*Scrollbar*height:                  15
*Scrollbar*shadowWidth:             2
*Scrollbar*cursorName:              top_left_arrow
*Scrollbar*pushThumb:               false

*shapeStyle:                        Rectangle
*beNiceToColormap:                  False
*shadowWidth:                       2
*SmeBSB*shadowWidth:                2
*highlightThickness:                0
*topShadowContrast:                 20
*bottomShadowContrast:              55

emacs*geometry:                     100x70-1+1
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Los clientes y los recursos poseen una clase genérica a la que pertenecen y un nombre que los identifica dentro de la clase (a menudo se usa el mismo término comenzando por mayúsculas para la clase y por minúsculas para el nombre): los clientes pueden además referenciarse por un nombre único para cada instancia. Esta jerarquía permite establecer diferentes alcances para una determinada asignación, tanto para la parte de selección de cliente como para la de selección de recurso. Así, de mayor a menor generalidad, una modificación puede afectar a todos los clientes, a una clase de clientes, a un cliente o a una ejecución de un cliente. Para los recursos es posible referenciar una clase de recursos completa o un recurso determinado. En caso de conflicto, las asignaciones específicas tienen prioridad sobre las genéricas.

1.4.1 Opciones en línea de comandos

Al ejecutar un cliente X, es posible especificar una serie de opciones que afectan tanto a su apariencia como a su comportamiento. Algunos ejemplos son:

`-display` para indicar el servidor X que se quiere utilizar.

`-bd`, `-fg` o `-bg` para fijar los colores de bordes, primer plano o fondo de la ventana.

`-name` para asignar un nombre a la instancia del cliente que se va a ejecutar.

`-title` para establecer el título de la ventana de la aplicación.

`-xrdb` para incluir cualquier otra opción.

Ejemplo: Algunos clientes ejecutados con opciones en línea de comandos.

```
xclock -display nombre_display_destino
xterm -fg color_primer_plano -bg color_fondo
xeyes -title nombre_título
```

1.4.2 Uso del cliente `xrdb`

El cliente `xrdb` es ejecutado normalmente desde los programas de inicialización (`xsession` o `xinitrc`) para cargar los valores de los recursos establecidos por defecto. Posteriormente se puede utilizar para consultar la base de datos de recursos o modificarla.

Algunas opciones admitidas por este comando son:

`-merge` para añadir los nuevos recursos a la base de datos.

`-query` para realizar consultas.

`-remove` para eliminar los recursos indicados.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

`-load` para cargar la base de recursos con los nuevos datos, descartando los valores anteriores.

Ejemplo: Diferentes usos del cliente `xrdb`.

`xrdb -merge nombre_fichero_recursos` (añade los recursos incluidos en el fichero a la base de datos)

`xrdb *Foreground: red` (reemplaza la base de datos por la especificación de recurso indicada)

`xrdb -query` (lista todos los recursos de la base de datos)

1.4.3 Recursos especiales

Algunos recursos poseen características especiales. El color, por ejemplo, se establece en base a un fichero denominado `rgb.txt`, en el directorio del sistema. Este fichero contiene diferentes nombres de colores definidos en base a sus componentes RGB. Alternativamente, se puede especificar el color de forma directa en el recurso a partir de las componentes RGB, empleando de uno a cuatro dígitos hexadecimales para cada una precedidos de un signo "#".

El tamaño de una ventana y su localización pueden especificarse a partir del recurso `geometry`. El formato a emplear es

`AnchoxAlto{+|-}Columna{+|-}Fila`

donde el ancho y el alto vienen dados en caracteres o pixels dependiendo del tipo de cliente, y los signos indican desplazamientos en pixels de una parte del marco (+ superior o izquierdo, - inferior o derecho) con respecto al borde de la pantalla correspondiente.

Ejemplo: Diferentes ubicaciones y tamaños empleando el recurso `geometry` en línea de comandos.

`xterm -geometry 80x24+0+0` (extremo superior izquierdo)

`xload -geometry 48x48+0-0` (extremo inferior izquierdo)

Las fuentes tienen asociadas cuatro recursos fundamentales que son `Font` para la fuente genérica del usuario, `FontList` para las áreas del sistema de los clientes, y `XmText*FontList` o `XmTextField*FontList` para las áreas de texto de los clientes. El siguiente subapartado está dedicado a la administración de las fuentes en X.

Ejemplo: Especificación de fuente genérica de usuario en fichero de recursos para todos los clientes de la clase `Xterm`.

```
Xterm*Font:  6x13bold
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

1.4.4 Fuentes en X

Una fuente es un estilo de texto que se emplea para imprimir caracteres. Existen dos tipos de fuentes en X que son las de mapa de bits, almacenadas como matrices de puntos, y las escalables, de las que se almacena una descripción matemática.

Las fuentes se designan en base al convenio *X Logical Font Description* (XLFD), que establece un formato de 15 campos para el nombre. Estos campos contienen diversos datos relativos a las características de la fuente como son sus dimensiones, la orientación, la familia o tipo base, etc.

Ejemplo: El siguiente nombre de fuente

```
-sony-fixed-medium-r-normal--16-120-100-100-c-80-iso8859-1
```

indica entre otras cosas un tipo de letra perteneciente a la familia fixed, de grosor medio, recta y de 12 puntos de altura a una resolución de 100x100 puntos por pulgada.

Las fuentes se almacenan en diferentes directorios del sistema llamados directorios de fuentes, que normalmente se ubican bajo `/usr/lib/X11/fonts`. Dentro de cada uno de estos directorios se encuentran los ficheros asociados a cada fuente además de dos herramientas especiales, el `fonts.dir`, que lista los nombres de todas las fuentes disponibles, y `fonts.alias`, que especifica nombres abreviados para referenciar con comodidad a las fuentes más utilizadas.

La instalación de una nueva fuente requiere un proceso que depende de su tipo, si bien puede resumirse en los siguientes pasos:

1. Creación de los directorios y copia de los ficheros.
2. Cambio de formato (mapa de bits y escalables) / Carga del conjunto de caracteres (escalables).
3. Actualización del contenido de los directorios y de las rutas de búsqueda.
4. Adición de licencia (escalables).

La variable de entorno FONTPATH contiene los nombres de los directorios de fuentes disponibles. Esta variable es fijada inicialmente desde el fichero de configuración hardware, pudiendo modificarse posteriormente por medio del cliente `xset`.

1.5 El manejador de ventanas

El gestor o manejador de ventanas es un cliente X especial encargado de organizar el comportamiento de las ventanas de las aplicaciones sobre el display. Es el responsable de determinar la ubicación de las ventanas, sus cambios de posición o tamaño, modos de desplazamiento, iconificación, políticas de enfoque, etc. El manejador de ventanas asocia

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

a cada ventana de las aplicaciones X en ejecución un marco con diferentes zonas o regiones activas, como son el título, los botones de menú, maximización y minimización, esquinas y bordes. Para estos elementos pueden definirse diferentes comportamientos mediante las opciones de configuración del manejador.

Existen diversos manejadores de ventanas disponibles, cada uno con un modo de funcionamiento y apariencia (*"look and feel"*) característicos. En principio, se suministran varios gestores estándar como `twm` o `fvwm` con la distribución de X, aunque luego cada fabricante de software suele proporcionar el suyo propio.

El manejador de ventanas se lanza al arrancar la sesión X desde los ficheros de inicialización correspondientes, dependiendo de que se haya realizado un inicio manual (`xinitrc`) o automático (`xsession`). Habitualmente es el último cliente que se ejecuta, por lo que la sesión está vinculada al mismo, finalizando ésta cuando el manejador se cierra.

1.5.1 Configuración general

En este apartado nos apoyaremos en un manejador en concreto, el `fvwm`, para presentar las diferentes opciones de configuración que admiten estos clientes.

El `fvwm` o gestor virtual de ventanas incorpora características como representación de escritorios virtuales (con un paginador para su visualización global), posibilidad de definir menús desplegables, módulos de utilidades predefinidos, etc. Las opciones de configuración son leídas por el manejador durante su inicio desde el fichero del usuario `.fvwmrc` o, en su defecto, desde el fichero del sistema `system.fvwmrc`.

Los comandos de configuración son de muy diverso tipo, apareciendo en el fichero de configuración en líneas separadas. Los comandos que afectan a los colores tienen el formato

```
identificador_objeto[Back/Fore]Color nombre_color
```

donde `identificador_objeto` determina el elemento al que se le va a aplicar el comando, como ventanas activas (`Hi`), menús (`Menu`), paginador (`Pager`). etc.

Ejemplo: Asignación de colores dentro del fichero de configuración `.fvwmrc`.

```
StdForeColor    Black
StdBackColor    LightSkyBlue
HiForeColor     yellow
HiBackColor     PeachPuff1
PagerBackColor  BlanchedAlmond
PagerForeColor  orchid
StickyForeColor Black
StickyBackColor #60c0a0

MenuForeColor  Black
MenuBackColor  grey
MenuStippleColor SlateGrey
```

Los comandos que afectan a las fuentes son de la forma

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
identificador_objetoFont nombre_fuente
```

y pueden aplicarse a ventanas (`Window`), iconos (`Icon`), etc.

Ejemplo: Establecimiento de las fuentes en `.fvwmrc`.

```
Font       -adobe-helvetica-medium-r-*-*-14-*-*-*-*-*-*-*
WindowFont -adobe-helvetica-bold-r-*-*-12-*-*-*-*-*-*-*
IconFont   fixed
```

Otros comandos toman la forma de propiedades que se aplican sobre aplicaciones o clases identificadas por un nombre de ventana, como es el caso de `NoTitle`, `NoBorder`, `Sticky`, `StaysOnTop`, etc. Pueden combinarse varios de estos comandos en una misma línea con ayuda del comando `Style`, de la forma

```
Style "nombre_cliente"Propiedad1, Propiedad2, ...
```

Ejemplo: Uso del comando `Style`.

```
Style   "*"             BorderWidth 5, HandleWidth 5, Color Black #60a0c0,
                          Icon unknown1.xpm
Style   "Fvwm*"         NoTitle, Sticky, WindowListSkip
Style   "Fvwm Pager"    StaysOnTop
Style   "FvwmBanner"    StaysOnTop
Style   "GoodStuff"     NoTitle, NoHandles, Sticky, WindowListSkip,
                          BorderWidth 0
Style   "*lock"         NoTitle, NoHandles, Sticky, WindowListSkip
Style   "xbiff"         NoTitle, Sticky, WindowListSkip
Style   "Maker"         StartsOnDesk 1
Style   "matlab"        StartsOnDesk 3
Style   "signal"        StartsOnDesk 3
Style   "rxvt"          Icon  term.xpm
Style   "xterm"         Icon xterm.xpm, Color black/grey
Style   "Appointment"   Icon datebook.xpm
Style   "xcalc"         Icon xcalc.xpm
Style   "xbiff"         Icon mail1.xpm
Style   "xmh"           Icon mail1.xpm, StartsOnDesk 2
Style   "xman"          Icon xman.xpm
Style   "xvgr"          Icon graphs.xpm
Style   "matlab"        Icon math4.xpm
Style   "xmag"          Icon mag-glass.xpm
Style   "xgraph"        Icon graphs.xpm
Style   "GoodStuff"     Icon toolbox.xpm
```

El escritorio puede configurarse a través del uso de comandos como `DeskTopSize` o `DeskTopScale`. Por otra parte, el comportamiento del entorno ante las acciones del ratón puede alterarse mediante comandos del tipo `OpaqueMove`, `ClickToFocus`, `EdgeResistance`, etc. También es posible definir hasta cuatro áreas para contener iconos empleando el comando `IconBox`.

Ejemplo: Diversas opciones de configuración para el manejador de ventanas `fvwm`.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
AutoRaise       750
ClickToFocus

IconBox         -150 90 -5 -140
IconBox         5 -140 -140 -5
IconBox         -70 1 -1 -140
StubbornIcons
StickyIcons

OpaqueMove      100
EdgeScroll      100 100
EdgeResistance  10000 0

RandomPlacement
SmartPlacement

DeskTopSize     3x3
DeskTopScale    36
Pager           -5 -5
```

1.5.2 Menús desplegables, funciones y módulos

Los menús desplegables permiten al usuario del manejador de ventanas organizar diferentes herramientas y utilidades del sistema para acceder a las mismas de forma estructurada. La forma de de definirlos es la siguiente

```
Popup ''nombre-menú''
   Función1 ''Etiqueta1'' opciones1
   Función2 ''Etiqueta2'' opciones2
   ...
EndMenu
```

donde las funciones que aparecen en cada entrada del menú están dentro de un conjunto de funciones predefinidas en el sistema. Algunos ejemplos de dichas funciones básicas son:

`Title` para especificar la cadena que identifica al menú.

`Nop` que no realiza ninguna acción y se utiliza para establecer separadores.

`Beep` que emite un sonido.

`Quit` para finalizar la ejecución del manejador de ventanas.

`Destroy` que cierra la aplicación vinculada al menú.

`Popup` para hacer referencia a un submenú previamente definido.

`Function` que permite invocar funciones complejas previamente definidas.

`Exec` para ejecutar cualquier comando.

Ejemplo: Menús desplegables.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
Popup "Shells"
    Title "Shells"
    Exec  "Xterm (7x14 f.)"          exec xterm -sb -sl 500 -j -ls -fn 7x14 &
    Exec  "Color Rxvt (VT100 em.)" exec rxvt -font 7x14 -ls &
    Exec  "C.Xterm (7x14 f.)" exec color-xterm -sb -sl 500 -j -ls -fn 7x14 &
    Exec  "Large Xterm (10x20 f.)" exec xterm -sb -sl 500 -j -ls -fn 10x20 &
EndPopup

Popup "Screensaver"
    Title  "Screensaver"
    Exec   "Bat"           exec xlock -nolock -nice 0 -mode bat &
    Exec   "Blank"         exec xlock -nolock -nice 0 -mode blank &
    Exec   "Life3d"        exec xlock -nolock -nice 0 -mode life3d &
EndPopup

Popup "Applications"
    Title  "Applications"
    Exec   "Emacs"         exec emacs -geometry 80x24-1+1 -fn 7x14 &
    Exec   "XV"            exec xv &
    Exec   "XFig"          exec xfig &
    Exec   "Ghostview"     exec ghostview &
EndPopup
```

El símbolo & introducido en la etiqueta de una entrada del menú hace que el caracter que venga a continuación aparezca resaltado, lo que indica que la opción correspondiente es seleccionable a través del teclado.

Una vez definido, un menú desplegable puede incorporarse como submenú a otro empleando la función Popup.

Ejemplo: Menú desplegable con submenús.

```
Popup "Utilities"
    Title  "Utilities"
    Exec   "Xterm"         exec color-xterm -sb -sl 500 -j -ls -fn 7x14 &
    Exec   "Emacs"         exec emacs -geometry 80x24-1+1 -fn 7x14 &
    Nop    ""
    Popup  "Applications" Applications
    Nop    ""
    Popup  "Shells"       Shells
    Nop    ""
    Popup  "Screensaver"  Screensaver
    Nop    ""
    Quit   "Quit"
EndPopup
```

Es posible definir funciones complejas dentro del manejador de ventanas fvwm a partir de las funciones básicas. Para ello se emplea la construcción

```
Function ''nombre-función''
    Función1 ''Etiqueta1'' opciones1
    Función2 ''Etiqueta2'' opciones2
    ...
EndFunction
```

Ejemplo: Construcción de funciones.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
Function "Move-or-Raise"
    Move       "Motion"
    Raise      "Motion"
    Raise      "Click"
    RaiseLower "DoubleClick"
EndFunction

Function "Maximize-func"
    Maximize "Motion" 0 100
    Maximize "Click" 0 80
    Maximize "DoubleClick" 100 100
EndFunction

Function "Resize-or-Raise"
    Resize     "Motion"
    Raise      "Motion"
    Raise      "Click"
    RaiseLower "DoubleClick"
EndFunction
```

Las funciones complejas construidas pueden entonces ser introducidas en menús desplegables con ayuda de la función predefinida `Function`.

Ejemplo: Menú desplegable con funciones complejas.

```
Popup "Window Ops"
    Title    "Window Ops"
    Function "Move"          Move-or-Raise
    Function "Resize"        Resize-or-Raise
    Raise    "Raise"
    Lower    "Lower"
    Iconify  "(De)Iconify"
    Stick    "(Un)Stick"
    Function "(Un)Maximize" Maximize-func
    Nop      ""
    Destroy  "Destroy"
    Delete   "Delete"
    Nop      ""
    Refresh  "Refresh Screen"
EndPopup
```

Los menús desplegables y las funciones también pueden asociarse a determinadas pulsaciones del ratón o del teclado. La sintaxis de estos comandos de enlace es como sigue

```
Mouse número_botón opciones [Popup "nombre_menú Function "nombre_función"]
      Key identificador_tecla opciones [Popup "nombre_menú Function
                                  "nombre_función"]
```

donde `opciones` permite fijar el entorno de operación del enlace mediante claves de contexto y de modificador. Así, para el contexto existen claves para indicar cualquier región (`A`), ventana raíz (`R`), botones izquierdo, derecho o extremo derecho de la barra de título (1,2 o 4), título (`T`), esquinas o laterales (`TS`), icono (`I`), ventana (`W`), etc. Para el modificador se puede indicar cualquiera (`A`), `Alt` (`M`), `Ctrl` (`C`), `Shift` (`S`), etc.

Ejemplo: Asignación de menús y funciones a pulsaciones de ratón y teclado.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
Mouse 1     R A     PopUp       "Utilities"
Mouse 2     R A     PopUp       "Window Ops"

Mouse 0     2 A     Function    "maximize-func"
Mouse 0     4 A     Iconify

Mouse 1     F A     Function    "Resize-or-Raise"
Mouse 1     T SA    Function    "Move-or-Raise"

Key Left    A C     Scroll      -100 +0
Key Right   A C     Scroll      +100 +0
Key Up      A C     Scroll      +0   -100
Key Down    A C     Scroll      +0   +100

Key F1      A M     Popup       "Utilities"
Key F2      A M     Popup       "Window Ops"
Key F4      A M     Iconify
```

El manejador `fvwm` permite además la definición de "cajas de herramientas" o módulos para acelerar el acceso a aplicaciones y utilidades de frecuente uso a través de barras de botones. Los módulos se definen en los menús desplegables por medio de la función predefinida `Module`. Posteriormente se fijan recursos para establecer el título del módulo, el número de columnas o filas de la barra de botones, elementos que lo integran, etc.

Ejemplo: Definición de módulo y configuración.

```
Popup "Module-Popup"
  Title  "Modules"
  Module "GoodStuff" GoodStuff
  ...
EndPopup

Popup "Utilities"
  Title  "Utilities"
  ...
  Popup  "Modules" Module-Popup
  ...
EndPopup

*GoodStuffFore      Black
*GoodStuffBack      #908090
*GoodStuffFont      -adobe-helvetica-bold-r-*-*-10-*-*-*-*-*-*-*
*GoodStuffGeometry -1-90

*GoodStuffColumns  1

*GoodStuff Kill     rbomb.xpm     Destroy
*GoodStuff Xcalc    rcalc.xpm     Exec "Calculator" xcalc &
*GoodStuff mail     mail2.xpm     Exec "xmh" xmh &
...
```

1.6 Clientes especiales y utilidades X

Existen, además del manejador de display (`xdm`) y el manejador de ventanas, otros clientes que desempeñan tareas importantes dentro del sistema. La mayoría ya han sido presentados en secciones anteriores. No obstante, se incluye a continuación un listado resumen de diferentes clientes especiales y la función que realizan.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

`xmodmap`: permite modificar el mapeo entre teclas y símbolos.

`xset`: ajusta diferentes opciones de display.

`xrdb`: gestiona la base de datos de recursos.

`xwd`, `xwud`, `xpr`: permiten almacenar, recuperar e imprimir una ventana.

`xfd`: visualiza una determinada fuente.

`xauth`: gestiona la información de seguridad.

`xhost`: organiza la lista de control de acceso al servidor X.

El usuario dispone además de múltiples utilidades de propósito general desarrolladas para este entorno. Sirvan de ejemplo las siguientes:

`xload`: visualización de la carga del equipo.

`xcalc`: calculadora.

`xclock`: reloj.

`xv`: permite la visualización y manipulación de imágenes en diferentes formatos.

`emacs`: editor de textos.

`ghostview`: visualiza documentos postscript.

`xman`: consulta de páginas de manual.

`xfig`: para generar figuras.

`xpaint`: para generar imágenes.

`nestcape`: navegante para internet.

1.7 Introducción a la programación en X

La programación en X se basa en el intercambio de información entre el cliente y el servidor en base al protocolo X. La generación de código a este nivel, sin embargo, es sumamente tediosa, lo que ha llevado a definir una jerarquía de niveles destinados a simplificar la programación de las aplicaciones. Cada nivel se apoya en las funciones del nivel inferior para construir un conjunto más elaborado, por lo que se hace imprescindible tener una idea general de cómo funciona cada nivel si se desea conocer cómo trabaja X.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

1.7.1 Niveles de software

Comenzando por el nivel más elemental, la arquitectura de software X comprende los siguientes elementos:

- Protocolo X.
- Librerías X o *Xlib*.
- Intrínsecos X o *Xtoolkit*.
- Herramienta X.

El nivel Xlib es en realidad el primero disponible para su utilización desde la aplicación, puesto que proporciona la misma potencia que el protocolo X pero condensada en una serie de funciones más cómodas de utilizar. El siguiente capítulo está dedicado a la programación de las aplicaciones utilizando recursos de este nivel.

El nivel Xtoolkit permite a los programadores disponer de bloques unificados de elementos gráficos, denominados *Widgets*, como son menús, botones, cuadros de diálogo, etc. Con esto se consigue simplificar la tarea de programación y, al mismo tiempo, facilitar la interpretación de la aplicación estandarizando su aspecto.

Las herramientas constituyen el nivel más alto en la programación X, y consisten en extensiones de los intrínsecos desarrolladas por diferentes casas de software para su comercialización. Las utilidades disponibles con cada herramienta dan lugar a un conjunto homogéneo de aplicaciones con apariencia similar, con el objeto de reducir tanto los tiempos de desarrollo de aplicaciones como los de aprendizaje de uso. Los diferentes fabricantes comercializan además su propio manejador de ventanas junto con la herramienta.

La herramienta propietaria más utilizada en el desarrollo de aplicaciones X en la actualidad es *OSF/Motif*, o simplemente Motif, de la Open Software Foundation, que es un consorcio en el que participan empresas como IBM, DEC o HewlettPackard. En el capítulo 3 se tratará en detalle la programación de las aplicaciones X basada en esta herramienta. El gestor de ventanas asociado a la herramienta Motif es el `mwm`.

Otras herramientas disponibles son *Open Look*, principal competidor de Motif, de AT&T y Sun Microsystems, o *DESQview/X* de Quarterdeck Office Systems.

En programación X se combinarán recursos procedentes de diferentes niveles. En cada punto a implementar hay que llegar a un equilibrio entre flexibilidad, que es máxima utilizando Xlib, y comodidad, que mejora con el uso de las herramientas. Por otra parte, y desde el punto de vista de la transportabilidad, a medida que se emplean recursos especializados de más alto nivel se hace más probable que la aplicación requiera modificaciones para su migración a otras plataformas.

1.7.2 Protocolo X

El protocolo X es el mecanismo básico de comunicación a través de la red en el entorno X. En base a este protocolo el cliente y el servidor intercambian datos de forma bidireccional

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

mediante mensajes. El protocolo X está soportado por otros protocolos de red de bajo nivel que admiten comunicación bidireccional tales como TCP/IP, DECnet o STREAMS.

Los mensajes del protocolo X se construyen a partir de los siguientes posibles formatos: *formato petición*, *formato respuesta*, *formato error* o *formato suceso*. El formato petición es utilizado por las aplicaciones para solicitar al servidor un servicio determinado, e incluye un campo de código de servicio, un campo de longitud y un campo de datos. Se trata de mensajes que normalmente no requieren respuesta y que se transmiten a través de colas, por lo que no bloquean la ejecución de la aplicación.

El formato respuesta se emplea en mensajes que se envían desde el servidor a las aplicaciones en respuesta a una determinada solicitud. La ejecución de la aplicación queda detenida desde el momento en que envía la petición hasta que recibe la respuesta desde el servidor. El mensaje de respuesta contiene campos de longitud y datos, así como identificadores de la petición asociada.

Cuando una aplicación produce un error en el sistema se le envía desde el servidor un mensaje con formato de error. Dichos mensajes poseen longitud fija y contienen información relativa al tipo de error detectado y la petición que lo ha desencadenado.

Por último, el formato suceso es empleado en mensajes generados a partir de la ocurrencia de un suceso o evento determinado, los cuales están asociados con actividades del sistema de entrada/salida. Estos mensajes tienen longitud fija y transportan información relativa al tipo de suceso y su identificación.

Los eventos constituyen la base a partir de la cual se define la funcionalidad de toda aplicación X. Existen 33 tipos distintos de eventos reconocidos en el sistema X, los cuales se pueden agrupar en diferentes categorías. Algunos ejemplos son eventos de teclado, de puntero, de foco de entrada, de exposición, de cambio de dimensiones, etc.

1.7.3 Programa básico

La programación en X, al menos para el nivel básico de funcionamiento, es eminentemente dirigida por eventos. Un programa genérico comprende, en líneas generales, las siguientes secciones:

- Inicialización: ficheros cabecera, constantes, variables globales, etc.
- Conexión: establecimiento del enlace cliente-servidor.
- Creación y activación de los elementos gráficos: ventana, menús, cuadros de diálogo, etc.
- Programación de eventos: definición de los tipos de mensajes de sucesos a procesar.
- Bucle de eventos: bucle infinito en espera a que se produzcan los eventos.

En el próximo capítulo se tratará en detalle la programación, presentando los diferentes recursos disponibles a través de ejemplos.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Capítulo 2

Introducción a la programación en X Window

2.1 Introducción

En este capítulo trataremos de ofrecer una visión somera y en lo posible completa de la programación en *X Window*. Es programación a bajo nivel, que como ya sabemos por el capítulo anterior, se ha visto facilitada de cara al programador de aplicaciones con diversas extensiones que simplifican la programación/descripción de la interfaz, a la vez que oculta parte de la flexibilidad que el X ofrece, por ello es interesante conocer la programación a más bajo nivel, en parte porque ofrece aspectos no tratados en las extensiones, y en parte porque nos hará entender la necesidad de dichas extensiones.

2.2 Un primer programa

Quizás al aprender a programar en C el primer trozo de código comprendido simplemente presentaba por pantalla un mensaje del estilo de 'Hola mundo', el típico primer programa en cualquier lenguaje estandarizado por Kernighan y Ritchie a partir de la aparición de C. Siguiendo esta pauta observaremos la estructura de un programa en X Window abordando este problema pero trasladado a nuestro entorno de ventanas, es decir, pensando que el mensaje va a aparecer en una ventana. Veamos a continuación el programa acompañado de los comentarios oportunos. Señalar que ejemplos similares pueden consultarse en [5] y [4].
Un primer punto del código es la especificación de los ficheros de cabecera que nos incluirán los prototipos de las funciones de Xlib, y los tipos específicos de variables de Xlib.

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Las declaraciones de variables, se observan tipos de variables no presentes en C estándar.

```
Display *dpy;
Window win;
int scr;
GC gc;
unsigned long fore,back;
XSizeHints thewin;
XEvent ev;
```

Se establece la conexión con el servidor (devuelve `NULL` en caso de error), pasando como parámetro el nombre del servidor al que deseamos conectarnos, en caso de pasar `NULL`, como ocurre en este caso, se acude a la variable `DISPLAY` del entorno que sigue el formato esperado `host:server.screen`. Las variables de entorno se pueden consultar con el comando `set`.

```
dpy=XOpenDisplay("");
```

Se recogen parámetros relativos a la conexión utilizados porteriormente en el resto del programa. Generalmente se utilizan macros para acceder a la estructura dpy, que se pretenda sea opaca al programador, y de esta forma pueda cambiarse el diseño de dicha estructura sin afectar a la programación.

```
scr=DefaultScreen(dpy);
```

Las aplicaciones no eligen los valores de los pixels sino que indicando un nombre de color mediante una macro obtenemos el índice de color más parecido contenido en el mapa de colores actual (ver apartado 2.6).

```
fore=WhitePixel(dpy,scr);
back=BlackPixel(dpy,scr);
```

Creamos una ventana con unas dimensiones y colores. La creación no implica que la ventana se presente en pantalla. La ventana, que es nuestro cliente, nuestra aplicación, se crea con sus propiedades esenciales que son las contenidas en la estructura de tipo `XSizeHints`.

```
thewin.height=200;
thewin.width=200;
win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
```

Establece las propiedades mínimas de la ventana, para que el gestor de ventanas (`window manager`) las tenga presentes.

```
XSetStandardProperties(dpy,win,"Ventana hola","Hola",None,
argv,argc,&thewin);
```

Crea el contexto gráfico, el GC, cuya función es definir la forma en la que se dibuja en una ventana (ver apartado 2.5.2).

```
gc = XCreateGC(dpy, win, 0, NULL);
XSetBackground(dpy,gc,back);
XSetForeground(dpy,gc,fore);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Seleccionamos los eventos que nuestra aplicación atenderá, en este caso la aplicación tan solo responderá ante un evento de exposición, `Expose`, situación que nos indica que el contenido de una ventana puede haberse perdido total o parcialmente, o que por primera vez se visualiza. Veremos que existe un amplio abanico de eventos, para responder al ratón, botones, teclas, etc.

```
XSelectInput(dpy,win,ExposureMask);
```

Presenta la ventana, la mapea en pantalla, lo que provocará un evento `Expose`, momento en el que ya podemos pensar en dibujar sobre la ventana, una vez que esta se ha presentado. En este ejemplo comprobaremos que por mucho que pasemos por encima nunca se borra. Para evitar los eventos múltiples se atiende en el bucle al campo `count` de la estructura, que será nulo con el último evento de una serie.

```
XMapWindow(dpy, win);
```

Una aplicación X es un programa dirigido por eventos, tras describir la interfaz y el comportamiento lanzamos el bucle de espera para que recoja los eventos que hemos indicado que esperamos a priori. Una vez descrita la ventana entramos en un bucle infinito de espera que responde a los eventos que previamente hemos especificado trataríamos. En este caso escribiremos un mensaje en la ventana. No hay puerta de salida del bucle por lo que el programa acaba al cerrar la ventana.

```
while (1) {
    XNextEvent(dpy,&ev);
    switch(ev.type){
      case Expose:
          if(ev.xexpose.count==0)
            DrawString(ev.xexpose.display,ev.xexpose.window,
            gc,80,95,"Hola Mundo",10);
      break;
       }
     }
XFreeGC(dpy,gc);
XDestroyWindow(dpy,win);
XCloseDisplay(dpy);
}
```

El programa ejemplo es bastante más extenso que la versión de C, y necesita de complicados comandos de compilación, motivo por el cual se utilizan guiones (*scripts*) o el comando `make`. Siguiendo la segunda opción para compilar el ejemplo, `hola-xlib.c`, invocamos al comando `make` que por defecto toma su comportamiento del fichero `Makefile` donde se indican las dependencias entre los ficheros a compilar. Por hacer este fichero más fácilmente transportable, indicaremos la localización de las bibliotecas de funciones de X, el aspecto variable de un sistema a otro, utilizando unas variables que modificadas nos adaptarán al sistema en el que nos encontramos. Al mismo tiempo en el mismo se indica la jerarquía a seguir para obtener un binario en concreto, en este caso `hola-xlib`. Para este primer ejemplo el fichero utilizado en un sistema HP-UX es:

```
OBJS = hola-xlib.o
INCLUDE = -I/usr/include/X11R5
LIBRARIES = -L/usr/lib/X11R5 -lX11
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005



Figura 2.1: Una ventana

```
hola-xlib:   $(OBJS)
        cc -g $(OBJS) -o hola-xlib $(LIBRARIES)

hola-xlib.o: hola-xlib.c
        cc -c -g hola-xlib.c $(INCLUDE)
```

2.3 Conceptos y filosofía

El programa ejemplo comentado en el apartado previo, tiene una estructura que se repetirá constantemente en los programas que utilizan X Window, el principal define la interfaz y su comportamiento, y finaliza su código con un bucle infinito de espera que responde a todo evento que haya sido seleccionado. Este es un punto importante, se define un comportamiento ante la interacción hombre-máquina, pero será si acaso dentro del bucle de espera cuando se responde a la interacción, es decir, una vez que el comportamiento ha sido especificado. El resultado puede contemplarse en la figura 2.1.

Como ya sabemos por el capítulo anterior, el sistema X Window opera en base al esquema cliente-servidor. El cliente (nuestra aplicación) interactúa con el servidor (quien se encarga de la visualización) con un juego petición-respuesta a los servicios que el cliente solicita. Toda instrucción en Xlib de nuestro primer programa ejemplo, se traduce a un mensaje que se envía al servidor. Estos mensajes podrán requerir respuesta pero no es un requisito necesario, por ejemplo la petición de creación de una ventana no requiere respuesta.

Un servidor X dispone de objetos almacenados conocidos por el nombre de recursos, entre ellos podemos distinguir las ventanas, los contextos gráficos, los tipos de letra, los cursores, los mapas de colores y los de puntos. Cada uno de ellos será abordado más adelante.

El sistema X Window nos ofrece un camino para definir la interfaz de nuestra aplicación, su comportamiento viene dado en base a los eventos que se habilitan en ella y su posterior tratamiento. Toda actividad relacionada con la entrada o salida genera uno o una serie de eventos. Hay una amplia variedad de ellos, 33, y cada uno tiene asociada una estructra que encontramos en el fichero $< X11/Xlib.h >$, cada evento define la estructura

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

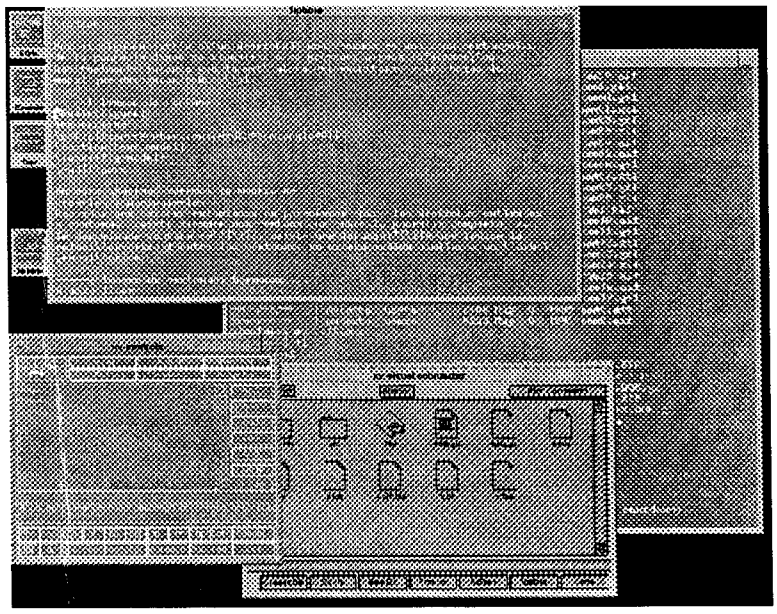

Figura 2.2: Una comunidad de ventanas

como más interesa para su posterior tratamiento. Tratamiento que tendrá lugar cuando a nuestra aplicación le indiquemos que esté atento a un determinado evento simplemente indicándole la máscara del evento en cuestión.

2.4 Las ventanas

Una ventana podemos definirla como una zona rectangular de la pantalla donde se visualiza o se presenta alguna información, habitualmente se asocia a la metáfora del escritorio y las hojas de papel que descansan sobre él.

En este apartado veremos la manera de crearlas y manipularlas, recordando que existen relaciones de parentesco entre ellas. Siempre existe una ventana que cubre la pantalla completa, es la denominada ventana raíz, de la que proviene el resto, se define por ello una jerarquía que indica la relación vertical entre las ventanas, en cierta medida similar a un árbol genealógico, por ello se llega a hablar de ventanas hermanas, ventanas apiladas, ocultas, etc.

En nuestro programa ejemplo al crear la ventana se indica la relación con una ventana padre `XDefaultRootWindow`, o como utilizamos en concreto en este caso una macro más óptima `DefaultRootWindow`.

2.4.1 Creación de ventanas

Con la llamada `XCreateSimpleWindow` se produce una petición de creación unidireccional, y se devuelve el identificador de la ventana para utilizar en el futuro, en este caso concreto la ventana no se visualiza, por lo tanto no oculta a ninguna otra, sus atributos se heredan del padre, que es identificado por el segundo parámetro de la llamada. En

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

nuestro programa ejemplo hacíamos uso de dicha llamada

```
Window XCreateSimpleWindow(dpy, padre,
x, y, ancho, alto, ancho_borde, pixel_borde,pixel_fondo);
Display *dpy; /* Identificador de la conexi\'on */
Window padre;   /* Identificador de la ventana padre,
                           en este caso la ra\'{\i}z */
int x,y; /* Coordenada esquina superior izquierda */
unsigned int ancho,alto;  /* Dimensiones */
unsigned int ancho_borde;  /* Ancho del borde */
unsigned int pixel_borde;  /* Valor del pixel del borde */
unsigned int pixel_fondo;  /* Valor del pixel del fondo */
```

El siguiente nivel de flexibilidad consiste en disponer de la posibilidad de aportar atributos a la ventana que creamos, para ello utilizamos `XCreateWindow`.

```
Window XCreateWindow(dpy, padre, x, y, ancho, alto,
ancho_borde, profundidad, clase, visual, mascara, atributos);
Display *dpy; /* Identificador de la conexi\'on */
Window padre;   /* Identificador de la ventana padre,
                           en este caso la ra\'{\i} */
intx,y; /* Coordenada esquina superior izquierda */
unsigned int ancho,alto;  /* Dimensiones */
unsigned int ancho_borde;  /* Ancho del borde */
unsigned int profundidad;  /* Profundidad de la ventana */
unsigned int clase;       /* Clase de ventana */
Visual *visual;      /* Tipo de visual */
unsigned long mascara;      /* Mascara de atributos a establecer */
XSetWindowAttributes *atributos; /* Atributos de la ventana */
```

Utilizando esta llamada con los valores de defecto, un ejemplo análogo a `hola-xlib.c`, es `pixmap-fondo.c` sólo que en lugar de dar un valor de pixel de fondo damos un mapa de pixels, en este caso el de la ventana padre, es decir, la raíz. El trozo de código modificado

```
...
XSetWindowAttributes   atributos;
...
atributos.background_pixmap=ParentRelative;
atributos.border_pixel=fore;
win = XCreateWindow(dpy, DefaultRootWindow(dpy),
 0, 0, thewin.width, thewin.height, 5,
 DefaultDepth(dpy,scr),InputOutput,DefaultVisual(dpy,scr),
     CWBackPixmap | CWBorderPixel, &atributos);
...
```

Indicar que antes de realizar la llamada proporcionamos los valores adecuados de los atributos utilizando una estructura de tipo `XSetWindowAttributes`, los campos a modificar los indicaremos en base a máscaras.

```
typedef struct{
 Pixmap background_pixmap; /* Mapa de pixel del fondo */
 unsigned long background_pixel;/* Valor del pixel del fondo */
 Pixmap border_pixmap; /* Mapa de pixel del borde */
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
int bit_gravity; /* Valor de atracci\'on del bit */
int win_gravity; /* Valor de atracci\'on de la ventana */
int backing_store; /* Tipo de preserva */
unsigned long backing_planes;  /* Planos a preservar */
unsigned long backing_pixel;   /* Valor del punto para restaurar */
Bool save_under; /* Guarda los bits de la ventana
                    por si hubiera un men\'u din\'amico */
long event_mask; /* Sucesos a atender */
long do_not_propagate_mask; /* Sucesos a no propagar */
Bool override_redirect; /* Tener o no en cuenta */
Colormap colormap; /* Mapa de colores */
Cursor cursor; /* Cursor */
}
```

Las máscaras mencionadas, que combinadas mediante una operación OR, indican el o los campos a modificar son las que siguen

```
#define   CWBackPixmap
#define   CWBackPixel
#define   CWBorderPixmap
#define   CWBorderPixel
#define   CWBitGravity
#define   CWWinGravity
#define   CWBackingStore
#define   CWBackingPlanes
#define   CWBackingPixel
#define   CWOverrideRedirect
#define   CWSaveUnder
#define   CWEventMask
#define   CWDontPropagate
#define   CWColormap
#define   CWCursor
```

2.4.2 Visualización de ventanas

En el paso anterior hemos creado la ventana, pero no la hemos visualizado. Para dicha tarea presentamos tres llamadas: `XMapWindow`, que nos presenta una ventana y sus descendientes que han solicitado ser visualizadas, generando con ello varios eventos. `XMapRaised` que produce el mismo efecto pero además eleva la ventana a la cima de la pila de ventanas. Y `XMapSubWindows` que presenta todas las subventanas de una dada. Indicar que para presentar una ventana se requiere una de estas llamadas, que sus ancestros sean visibles, que nadie la tape, y que la cola que contiene las peticiones realizadas al servidor se haya vaciado (`XFlush(dpy)`).

Recordar que una ventana puede no visualizarse sin necesidad de destruirla, para ello se utilizan las llamadas a `XUnmapWindow` y `XUnmapSubWindows`, que por analogía con el párrafo anterior harán invisibles la ventana y sus descendientes, o tan solo sus descendientes respectivamente. Si una ventana no se destruye, su estructura queda almacenada en memoria, y por consiguiente está creada y podremos volverla a visualizar cuando nos sea necesaria. Resulta sencillo construir un ejemplo que haga aparecer y desaparecer a una ventana jugando por ejemplo con `XMapWindow` y `XUnmapWindow`.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

2.4.3 Destrucción de ventanas

Las ventanas se destruyen implícitamente al destruir la conexión con el servidor (botón de la ventana), pero puede realizarse por código utilizando `XDestroyWindow` con lo que se destruye una ventana y sus descendientes, o `XDestroySubWindows` para ocuparnos sólo de los descendientes.

2.4.4 Aplicaciones con varias ventanas

Como hemos visto, el segundo parámetro de las llamadas para crear ventanas identifica la ventana padre de la que estamos creando, no hemos puesto restricciones sobre el número de ventanas a crear en una aplicación por lo que podemos plantearnos crear un aplicación con más de una ventana. A continuación mostramos dos ejemplos, uno de ellos con dos ventanas hermanas, `ventanas-hermanas.c`, y el otro con una ventana padre y otra hija, `ventanas-padre-hija.c`. Primero debemos declarar más variables para identificar ventanas

```
Window          win,win2;
XSizeHints      thewin,thewin2;
```

podemos probar a crear ventanas hermanas, en este caso ambas hijas de la ventana raíz

```
thewin.height=200;
thewin.width=200;
win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
  0, 0, thewin.width, thewin.height, 5, fore,back);
thewin2.height=100;
thewin2.width=100;
win2 = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
      0, 0, thewin2.width, thewin2.height, 5, fore,back);
```

o ventanas que son una hija de la anterior

```
thewin.height=200;
thewin.width=200;
win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
 0, 0, thewin.width, thewin.height, 5, fore,back);
  argv,argc,&thewin);
thewin2.height=100;
thewin2.width=100;
win2 = XCreateSimpleWindow(dpy, win,
     0, 0, thewin2.width, thewin2.height, 5, fore,back);
```

y establecer para ambas sus propiedades, visualizarla, y definir como antes los eventos que reconocen, como en este ejemplo.

```
XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
XSetStandardProperties(dpy,win2,"Ventana saludo II","Hola II",None,
    argv,argc,&thewin2);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

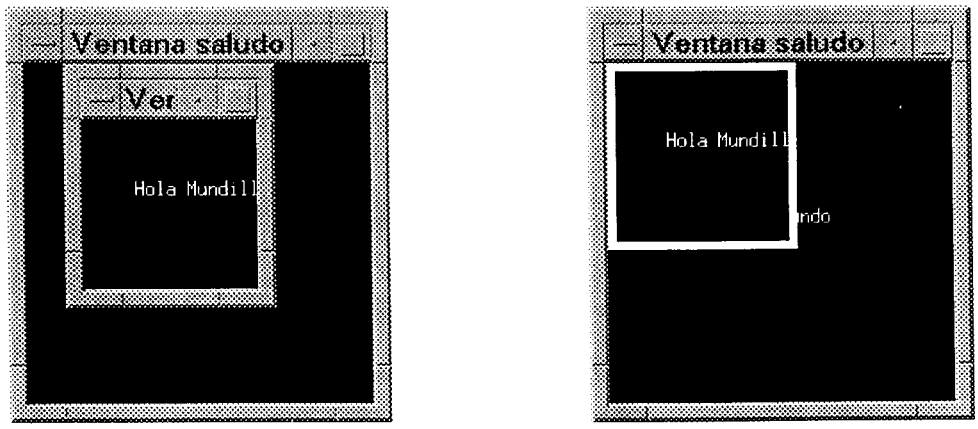

Figura 2.3: Dos ventanas hermanas a la izquierda, y padre e hija a la derecha

```
...
XSelectInput(dpy,win,ExposureMask | KeyPressMask);
XSelectInput(dpy,win2,ExposureMask);
XMapWindow(dpy, win);
XMapWindow(dpy, win2);
```

Los resultados pueden verse en la figura 2.3.

2.4.5 Geometría de una ventana

La ventana se define por su tamaño, en términos de su altura y anchura, por su anchura de borde, una posición dentro de la ventana padre y unas coordenadas que representarán el origen de la ventana, es decir, su ventana superior izquierda. Pueden obtenerse las dimensiones de la pantalla en pixels con `DisplayHeight` y `DisplayWidth`, o incluso con llamadas Xlib como `XGetGeometry` podemos recuperar dichos valores para la ventana raíz. En nuestro primer ejemplo todos esos valores, excepto la posición en el padre se explicitan en la llamada `XCreateSimpleWindow` , en siguientes apartados identificaremos todos estos parámetros.

2.4.6 Características propias de una ventana

Se denominan de esta manera aquellas características permanentes de las ventanas, las cuales se establecen en la creación. La llamada a `XCreateSimpleWindow` utiliza valores por defecto para las mismas, mientras que utilizando la llamada a una función más potente como es `XCreateWindow` podremos escoger dichas propiedades de la ventana.

En primer lugar nombraremos la clase de la ventana, esta podrá ser de tipo `InputOutput` o `InputOnly`, las primeras son las normales (las creadas por defecto) y las segundas son invisibles y utilizadas para gestionar la pantalla permitiendo eventos de ratón y teclado directos.

Otra propiedad es la profundidad de una ventana, que se corresponde con el número de bits por pixel (las ventanas `InputOnly`, dada su invisibilidad tienen profundidad nula).

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

El valor es dependiente de la estación de trabajo en la que se trabaje, en nuestro segundo ejemplo veremos que se toma dicha profundidad tras llamar a `XDefaultDepth`, lo cual implica que se esta tomando la misma profundidad que la ventana raíz. Un listado de las disponibles se obtiene con `XListDepths`. Por último nos queda hablar del `Visual`, esta propiedad define el modo en que la estación trabaja con la salida por pantalla, como por ejemplo el tema del color y el número de puntos por bit. Esperaremos al apartado de color y gráficos. El ejemplo `profundidad-visual.c` detecta para un servidor las profundidades posibles, y para cada una de ella los visuals disponibles.

```
...
int             *prof,i,j,cont,cont2;
XVisualInfo     *listavisual,template;
...
prof=XListDepths(dpy,scr,&cont);
 for (i=0;i<cont;i++){
  printf("Profundidad %d \n",prof[i]);
    template.screen=scr;
    template.depth=prof[i];
    listavisual=XGetVisualInfo(dpy, VisualDepthMask |
    VisualScreenMask, &template,&cont2);
    printf("Visuals %d\n",cont2);
    for (j=0;j<cont2;j++)
     {
     switch(listavisual[j].class){
       case PseudoColor:
        printf("PseudoColor \n");
       break;
       case StaticColor:
        printf("StaticColor \n");
       break;
       case DirectColor:
        printf("DirectColor \n");
       break;
       case TrueColor:
        printf("TrueColor \n");
       break;
       case GrayScale:
        printf("GrayScale \n");
       break;
        case StaticGray:
         printf("StaticGray \n");
        break;
   }
  }
}
```

2.4.7 Atributos de una ventana

Así se conocen a aquellas propiedades modificables de la ventana. Podemos comenzar mencionando los atributos del fondo, por un lado `background_pixel` genera un fondo continuo, con `background_pixmap` obtenemos un fondo como un mosaico obtenido a partir de un patrón, si su valor fuera `None`, que parece ser el valor por defecto al ejecutar `pixmap-fondo-None.c` , donde no se le da valor, veremos las ventanas ocultas. Otra alternativa para este campo es asociar el valor del padre, `ParentRelative`, como ocurre en `pixmap-fondo.c`, o a partir del formato `bitmap` crear el mapa de pixels o *pixmap* que

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005



Figura 2.4: Un bitmap para el fondo

asociamos a este campo, como en `pixmap-bitmap.c`. El borde tiene atributos análogos a los del fondo pero como es lógico sus nombres variarán ligeramente: `border_pixmap` y `border_pixel`. Escogiendo como ejemplo el aspecto del fondo, el trozo de código para asignar un mapa de pixels

/* Primero el bitmap */

```
#define dummy_width 40
#define dummy_height 40
 static char bits[] = {
  0x00, 0x80, 0x20, 0x20, 0x00, 0x40, 0x40, 0x20, 0x20, 0xc0, 0x41,
  0x40, 0x20, 0x20, 0x30, 0x46, 0x40, 0x20, 0x20, 0x08, 0x88, 0x20, 0x40,
  0x10, 0x08, 0x08, 0x11, 0x80, 0x0f, 0x04, 0x10, 0x0e, 0x00, 0x00, 0x04,
  0x10, 0x00, 0xfe, 0x00, 0x04, 0x10, 0x80, 0x01, 0x03, 0x08, 0x08, 0x40,
  0x00, 0x04, 0x38, 0x08, 0x20, 0x00, 0x08, 0xf6, 0x06, 0x20, 0x1f, 0x08,
  0xc1, 0x03, 0x10, 0x11, 0x10, 0x00, 0x02, 0x10, 0x11, 0x10, 0x00, 0x04,
  0x10, 0xfd, 0x1f, 0x00, 0x04, 0x90, 0x1f, 0xf0, 0x00, 0x08, 0x50, 0x00,
  0x10, 0x00, 0x08, 0x30, 0x00, 0x10, 0x00, 0x08, 0x18, 0x00, 0x10, 0x00,
  0x04, 0x24, 0x00, 0x08, 0x00, 0x04, 0x22, 0x00, 0x08, 0x00, 0x82, 0x4f,
  0x00, 0x04, 0x01, 0x43, 0x91, 0x01, 0x03, 0xc6, 0xa0, 0x20, 0xfe, 0x00,
  0x38, 0x90, 0x40, 0x00, 0x00, 0x00, 0x88, 0x80, 0x00, 0x00, 0x00, 0x48,
  0x80, 0x00, 0x00, 0x00, 0x48, 0x80, 0x00, 0x00, 0x3e, 0x48, 0x80, 0x00,
  0x00, 0x41, 0x48, 0x80, 0x00, 0x00, 0x80, 0x50, 0x40, 0x00, 0x00, 0x80,
  0x60, 0x20, 0x00, 0x00, 0x80, 0x40, 0x10, 0x00, 0x00, 0x80, 0xc0, 0x0f,
  0x00, 0x00, 0x80, 0x40, 0x00, 0x00, 0x00, 0x41, 0x40, 0x00, 0x00, 0x00,
  0x3e, 0x40, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x80,
  0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00};

  ...

 ima=XCreatePixmapFromBitmapData(dpy,DefaultRootWindow(dpy), bits,
 dummy_width, dummy_height, fore,back,DefaultDepth(dpy,scr));

 atributos.background_pixmap=ima;
 atributos.border_pixel=fore;

 win = XCreateWindow(dpy, DefaultRootWindow(dpy), 0, 0, thewin.width,
 thewin.height, 5, DefaultDepth(dpy,scr),InputOutput,
 DefaultVisual(dpy,scr), CWBackPixmap | CWBorderPixel, &atributos);
```

resultando la figura 2.4.

Al crear nuestras ventanas, seleccionamos los eventos que se van a tratar en su interior, en estos primeros ejemplos hemos utilizado el evento `Expose`, pero existen otros sencillos que nos sirven para comenzar a jugar con `Xlib`. Por ejemplo algunos eventos

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

rápidamente entendibles, como los de ratón `ButtonPress` y `ButtonRelease`, escogidos con las máscaras `ButtonPressMask` y `ButtonReleaseMask`, o un evento de teclado `KeyPress` cazado con `KeyPressMask`.

Asalta una cuestión, si a una ventana descendiente de otra no le activamos un evento de ratón, pero está activado en su padre, si ocurre ese evento dentro de la zona de la hija, esta no lo trata, pero como a su vez la hija puede estar en el interior de su padre el evento está ocurriendo también para el padre, por lo que si la hija no lo trata el evento se le envía al padre. Existe la posibilidad de ocultar estos eventos a la ventana padre utilizando en campo `do_not_propagate_mask` de la ventana hija, como en las siguientes líneas, `propagate.c`, en las que además el cambio se realiza con posterioridad a la creación.

```
XSetWindowAttributes   atributos;
...
atributos.do_not_propagate_mask=ButtonPressMask;
XChangeWindowAttributes(dpy,win2,CWDontPropagate,&atributos);
...
```

El campo `override_redirect` encarga al gestor de ventanas las tareas de visualización, configuración o de cambio de orden de ventanas. El mapa de colores debe ser del mismo tipo visual que la ventana, una opción es tomar el del padre. Con el atributo del cursor se especifica el cursor a usar cuando el puntero está en una ventana.

Un aspecto menos utilizado es seleccionar el comportamiento del contenido de la ventana cuando esta cambia de tamaño, el servidor sabe lo que hacer con el contenido de una ventana al moverla, pero no es lo mismo cuando se redimensiona, para indicar los atributos de reposicionamiento de una ventana respecto a su padre, y de la imagen de una ventana respecto a la ventana redimensionada son controlados respectivamente mediante `window_gravity` y `bit_gravity`. Con ellos se indica si la zona afectada es atraída por el centro, un borde o una esquina de la ventana, tomados como los ocho puntos de una brújula.

Cuando deseamos que sea el servidor X quien almacene el contenido de una ventana se utiliza una zona de memoria de reserva, `backing_store`, pero generalmente esta zona es muy pequeña para satisfacer las necesidades, razón por la cual esta responsabilidad se deja para los programadores, es decir, son ellos los que deben redibujar una zona de pantalla (además de evitar problemas de portabilidad). Tres valores son válidos para este campo, por defecto `NotUseful` indicando que el área de almacenamiento no se utiliza, con `WhenMapped` se utilizará solo si la ventana está visualizada, y con `Always` en todo momento. Para reducir el tamaño de la zona de almacenamiento se puede utilizar `backing_planes` para reducir el número de bits por pixel que se almacenan, y con `backing_pixel` asignamos valor a los planos restantes. Otro campo que controla cuando un menú diámico oscurece una ventana es `save_under`. Los dos aspectos anteriores no siempre están disponibles en los servidores X, motivo por el cual existen macros para consultar con el mismo las posibilidades que ofrece, tales macros son `DoesBackingStore`, que devuelve el valor disponible entre los tres posibles, y `DoesSaveUnders` que simplemente indica si es o no posible. En el ejemplo `backing_store.c` se consultan las posibilidades del servidor.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

2.4.8 Configurando ventanas

Podemos comenzar con acciones tales como desplazar o redimensionar una ventana, como norma general todas las llamadas requerirán el identificador de la conexión y el de la ventana.

Para desplazar la ventana además indicamos la posición de la esquina superior izquierda, todo con la llamada a `XMoveWindow`.

Para redimensionar indicamos los nuevos valores para el ancho y el alto con `XResizeWindow`.

Ambas tareas pueden realizarse simultáneamente con `XMoveResizeWindow`, por el nombre deducimos que primero se indican las coordenadas de la esquina.

La anchura del borde de la ventana se porporciona con `XSetWindowBorderWidth`.

Otro tema con posible modificación del orden de apilamiento de las ventanas. Una llamada a `XRaiseWindow` nos coloca una ventana en la cima, en el fondo se coloca con `XLowerWindow`.

También se puede controlar como circulan las ventanas por la pila, cuyo efecto se aprecia si las ventanas se ocultan unas a otras, la cima será aquella totalmente visible. Podremos rotar los hijos de una ventana especificada con `XCirculateSubwindows` pasando como parámetro la dirección (con `RaiseLowest` asciende a la cima la ventana actualmente más baja similar a `XCirculateSubwindowsUp`, con `LowerHighest` se desciende a la más alta análogo a `XCirculateSubwindowsDown`). Por último mencionar la llamada `XRestackWindows` que reordena la pila según el orden especificado como parámetro. Como es lógico en este caso en la llamada se pasa una lista de ventanas a ordenar y no tan solo una.

Algunos ejemplos pueden verse en `mueve.c`, `mueve_hija.c` y `tamaño.c`.

2.4.9 Modificando atributos

Comentábamos que los atributos son las características no permanentes de las ventanas. Descubrimos dos opciones a la hora de realizar la modificación, bien realizarlo de manera simultánea para varios atributos o bien de forma individual. En el primer caso a la llamada `XChangeWindowAttributes`, que mencionamos con anterioridad. Le pasaremos además del identificador de la conexión y la ventana, la máscara de atributos a establecer y los valores mediante la estructura ya mencionada `XSetWindowAttributes`. En el caso individual se utilizan las llamadas `XSetWindowBackground`, `XSetWindowBackgroundPixmap`, `XSetWindowBorder`, `XSetWindowBorderPixmap`, `XSetWindowColormap`, `XDefineCursor`. Como término medio tenemos para establecer la posición, el tamaño y la anchura del borde la llamada `XConfigureWindow`.

2.4.10 Eventos y ventanas

Se comentó la existencia de 33 eventos de ellos 10 se asocian a cambios de estado de las ventanas. Seguidamente los mencionamos, mas si se desea tener información sobre cuales son las llamadas que producen o pueden producir tal evento podemos remitir a [4].

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Cada evento tiene una estructura asociada con información de relevancia.

- `CirculateNotify`, informa sobre un cambio en la posición en la pila.
- `ConfigureNotify`, cambio en tamaño, posición, apilamiento o borde.
- `CreateNotify` y `DestroyNotify`, asociado a la creación y destrucción respectivamente.
- `GravityNotify`, una ventana se mueve por un cambio de tama·o del padre.
- `MapNotify`, `MappingNotify` y `MapNotify` se relacionan con el proceso de visualización y sus cambio.
- `ReparentNotify` cuando se produce un cambio de paternidad.
- `Visibility` y `Expose`, la primera informa sobre un cambio de visibilidad, mientras que la segunda indica que una ventana ha perdido total o parcialmente su contenido.

2.5 Gráficos, GC y Texto

2.5.1 Dibujos

Una vez que tenemos definido al menos un GC podemos comenzar a dibujar, en principio no nos ocupamos del significado de los campos de la estructura, que veremos en el siguiente apartado, sino que simplemente dibujamos sin preocuparnos. Las llamadas para dibujar son fáciles de usar, para dibujar un punto sólo necesitamos saber localizar donde va a ir el punto (servidor, ventana y coordenadas), y la estructura GC. Un listado de llamadas sin y con relleno

- `XDrawPoint`, dibuja un punto dadas sus coordenadas.
- `XDrawPoints`, dibuja un vector de puntos, se necesita el vector y el número de puntos que contiene.
- `XDrawLine`, una línea en base a dos puntos.
- `XDrawLines`, líneas entre puntos sucesivos de un vector.
- `XDrawSegments`, líneas entre pares de puntos de un vector.
- `XDrawRectangle`, dibuja un rectángulo dadas dos esquinas, la superior izquierda y la inferior derecha.
- `XDrawRectangles`, dibuja varios.
- `XDrawArc`, como `XDrawRectangle` pero pintando un arco que cabe en el rectángulo.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

- `XFillRectangle`, dibuja un rectángulo dadas dos esquinas, la superior izquierda y la inferior derecha.
- `XFillRectangles`, dibuja varios.
- `XFillArc`, como `XFillRectangle` pero pintando un arco que cabe en el rectángulo.
- `XFillArcs`, dibuja varios arcos.
- `XFillPolygon`, no existe su correspondiente para solo el contorno, dada una serie de puntos une los consecutivos, pero además cierra la serie uniendo el último con el primero. Se indica además la forma del polígono, `Complex`, `Convex` y `Nonconvex`, según ella se utilizarán rutinas más optimizadas o no. Indicando si las coordenadas se expresan absolutas o relativas al primero.

Para trabajar con mapas de pixels podemos utilizar llamadas para crearlos a partir de bitmaps con un formato como

```
#define gray_width 16
#define gray_height 16
#define gray_x_hot 8
#define gray_y_hot 8
static char gray_bits[] = {
    0xf8, 0x1f, 0xe3, 0xc7, 0xcf, 0xf3, 0x9f, 0xf9, 0xbf,
    0xfd, 0x33, 0xcc, 0x7f, 0xfe, 0x7f, 0xfe, 0x7e, 0x7e,
    0x7f, 0xfe, 0x37, 0xec, 0xbb, 0xdd, 0x9c, 0x39, 0xcf,
    0xf3, 0xe3, 0xc7, 0xf8, 0x1f};
```

todo ello con `XCreatePixmapFromBitmapData`.

Disponemos de llamadas para limpiar zonas de pantalla, aunque indicar que siempre se mantiene el aspecto de fondo establecido para la ventana. Podemos limpiar una ventana completa con `XClearWindow` y una zona con `XClearArea`. De igual forma podemos áreas de una zona a otra o incluso desde mapas de pixels que no están en pantalla con `XCopyArea`. Diferente es la actividad de copiar planos, con `XCopyPlane`.

Pensemos en dibujar rectas desde donde picamos (evento de ratón), al origen de la ventana, `rectas.c`.

```
...
XSelectInput(dpy,win,ButtonPressMask | KeyPressMask);
...
case ButtonPress:
 XDrawLine(ev.xexpose.display,ev.xexpose.window,gc,0,0,
 ev.xbutton.x,ev.xbutton.y);
break;
...
```

Recurriendo a los manuales podremos observar la estructura asociada a cada evento y para este caso recuperar la posición donde se produjo el evento de ratón.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

2.5.2 El Contexto Gráfico

Para dibujar gráficos se utilizan las llamadas primitivas gráficas, con ellas pintamos puntos, líneas, texto, imágenes, etc. Estas primitivas no contienen toda la información necesaria para dibujar, ello es posible por la existencia de un recurso, el contexto gráfico (GC en adelante), donde se almacenan variables aplicables a cada petición para dibujar. Todo excepto el borde y el tramado del fondo de una ventana está contemplado en este recurso.

Con el GC se reduce el tráfico con el servidor ya que es información que no circula con cada petición, es posible modificarlo, se pueden mantener varios, todo ello beneficiando la eficiencia, y permitiendo no tener que proporcionar en cada petición un innumerable número de parámetros que en realidad no tienen que ver con nuestra petición.

Creación y configuración

Antes de utilizarlo debemos crearlo con la llamada a `XCreateGC`, a la que se le pasan el identificador de la conexión con el servidor o *display*, el identificador de una ventana o un mapa de pixels, los valores en base a la estructura `XGCValues` y la máscara que indica los valores de la estructura anterior que se establecen en la creación. Si recordamos unas líneas de `hola-xlib.c`, en las que los valores se indicaban a posteriori para dos de sus campos.

```
...
gc = XCreateGC(dpy, win, 0, NULL);
XSetBackground(dpy,gc,back);
XSetForeground(dpy,gc,fore);
...
```

La estructura mencionada, con sus valores por defecto, es como sigue

```
typedef struct {
 int function;      /* funci\'on l\'ogica, por defecto GXcopy */
 unsigned long plane_mask;/* m\'ascara de planos, todo a 1
                     por defecto*/
 unsigned long foreground;/* valor pixel de dibujo,
                     por defecto 0 */
 unsigned long background;/* valor pixel fondo,
                     por defecto 1 */
 int line_width;    /* ancho l\'{\i}nea, 0 por defecto */
 int line_style;    /* LineSolid (defecto), LineOnOffDash,
                     LineDoubleDash */
 int cap_style;     /* CapNotLast, CapButt (defecto),
   CapRound, CapProjecting */
 int join_style;    /* JoinMiter (defecto), JoinRound,
   JoinBevel */
 int fill_style;    /* FillSolid (defecto), FillTiled,
   FillStippled */
 int fill_rule;     /* EvenOddRule (defecto), WindingRule */
 int arc_mode;      /* ArcPieSlice (defecto), ArcChord */
 Pixmap tile;       /* Mapa de pixels mara mosaico, por defecto
                     con pixel dibujo */
 Pixmap stipple;    /* Mapa de pixel de profundidad 1 */
 int ts_x_origin;   /* desplazamiento para los mosaicos,
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
                    0 por defecto */
  int ts_y_origin;
  Font font;        /* fuente de texto por defecto, depende de
                    la implementaci\'on */
  int subwindow_mode;/* ClipByChildren (defecto), IncludeInferiors */
  Bool graphics_exposures;/* se generan eventos para XCopyArea,
                    XCopyPlane. True */
  int clip_x_origin;/* origen para clipping, 0 por defecto */
  int clip_y_origin;
  Pixmap clip_mask; /* bitmap para clipping, 0 por defecto */
  int dash_offset;  /* informaci\'on para trazos */
  char dashes;
} XGCValues;
```

Cada campo de la estructura tiene una máscara asociada para indicar cuando estamos estableciendo su valor.

```
#define GCFunction
#define GCPlaneMask
#define GCForeground
#define GCBackground
#define GCLineWidth
#define GCLineStyle
#define GCCapStyle
#define GCJoinStyle
#define GCFillStyle
#define GCFillRule
#define GCTile
#define GCStipple
#define GCTileStipXOrigin
#define GCTileStipYOrigin
#define GCFont
#define GCSubwindowMode
#define GCGraphicsExposures
#define GCClipXOrigin
#define GCClipYOrigin
#define GCClipMask
#define GCDashOffset
#define GCDashList
#define GCArcMode
```

En el ejemplo `hola-xlib.c` se crea un GC sin establecer valores, es decir, todos los campos se establecen por defecto. Si deseamos establecer algún cambio debemos escoger los campos de la estructura, rellenarlos y realizar la llamada ya mencionada. De igual forma podemos crear dos o más GC's y simplemente al realizar las primitivas referenciar al que nos interese.

Como es obvio con el GC controlamos de forma exacta los pixels y planos afectados por las primitivas de dibujo. Cuando se realiza una petición el mapa de bits generado por ella obtiene los valores de los pixels en base a los campos `foreground` y `background` de la estructura GC. Luego se realiza una operación lógica entre este mapa y el valor de los pixels de la zona donde vamos a pintar o destino definido por `function`. A este resultado se le aplica sucesivamente la operación lógica AND con la máscara de planos `plane_mask`, que nos selecciona los planos que intervienen, y la de recorte, `clip_mask`, que selecciona los pixels que intervienen.

Los valores posibles para `function` que nos indica cómo se combinan los valores de los nuevos pixels, src, con los ya existentes en la zona de la pantalla, dst, pueden ser los siguientes.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
GXclear          0
GXand            src AND dst
GXandReverse     src AND (NOT dst)
GXcopy           src
GXandInverted    (NOT src) AND dst
GXnoop           dst
GXxor            src XOR dst
GXor             src OR dst
GXnor            (NOT src) AND (NOT dst)
GXequiv          (NOT src) XOR dst
GXinvert         (NOT dst)
GXorReverse      src OR (NOT dst)
GXcopyInverted   (NOT src)
GXorInverted     (NOT src) OR dst
GXnand           (NOT src) OR (NOT dst)
GXset            1
```

Un ejemplo puede ser `mueve_recta.c`, aqui con el botón izquierdo pintamos la recta que se une con el origen y al movernos con un botón pulsado, desplazamos el punto final de la recta. Para borrar la anterior se utilizan dos técnicas distintas, con el botón izquierdo borramos toda la ventana, ocurre que si hay muchos objetos al redibujarlos todos se puede percibir parpadeo. Con el central nos evitamos redibujar todo, pero podemos deteriorar otras figuras al pintar usando la máscara `GXxor`. Vemos el código en el que se introduce el tratamiento del evento de movimiento del puntero al pulsar un botón.

```
...
XSelectInput(dpy,win,ButtonPressMask | KeyPressMask | ButtonMotionMask);
...
  case ButtonPress:
    XDrawLine(ev.xexpose.display,ev.xexpose.window,gc,0,0,
    ev.xbutton.x,ev.xbutton.y);
  break;
  case MotionNotify:
    switch(ev.xbutton.button){
     case Button1:
      XClearWindow(dpy,win);
     break;
     case Button2:
      XSetFunction(dpy,gc,GXxor);
      XDrawLine(ev.xexpose.display,ev.xexpose.window,gc,0,0,x,y);
      x=ev.xbutton.x;
      y=ev.xbutton.y;
      XSetFunction(dpy,gc,GXcopy);
     break;
    }
    XDrawLine(ev.xexpose.display,ev.xexpose.window,gc,0,0,
    ev.xbutton.x,ev.xbutton.y);
  break;
...
```

Con el GC definimos todo aquello configurable a la hora de dibujar, tendremos por ejemplo el relleno de patrones, para las líneas podremos definir su grosor, su estilo (continuo, a trazos, doble trazos), la longitud y aspectos de los trazos, el aspecto de los extremos, de las uniones. Para ampliar detalles recomendamos [5] y [4].

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

2.5.3 Las fuentes y el texto

Bastante más complejo es el tema de las fuentes y el texto. Para empezar podemos decir que toda llamada para dibujar texto tiene dos versiones en función de si se manejan fuentes de 1 byte o de 2. Las primeras se limitan a 256 caracteres mientras que las segundas a 65536 (pensando en Oriente).

El primer paso antes de usar una fuente es cargarla, en realidad en los ejemplos hemos dibujado texto, pero estábamos utilizando la fuente por defecto. Varios clientes X pueden compartir la fuente, pero todos han obtenido previamente el identificador de la misma al cargarla. Se plantea la pregunta de saber las fuentes disponibles en el servidor, una lista se obtiene con `XListFonts` y `XListFontsWithInfo` (que no recoge la métrica de los caracteres), puede utilizar comodines al buscar como `*`. La lista puede eliminarse cuando ya no sea necesaria con `XFreeFontNames`. En `fuente.c` leemos 100 fuentes que encajen con `*bold*`

```
...
char    **fuentes;
int     cont;
Font    ft;

...
fuentes=XListFonts(dpy,"*bold*",100,&cont);
 for (i=0;i<cont;i++)
     printf("Tipo %s\n",fuentes[i]);

...
```

Con el nombre de la fuente ya podemos proceder a cargarla, `XLoadFont` nos devolverá un identificador que utilizaremos en sucesivas referencias a la fuente, por ejemplo podemos asignar la fuente al GC con `XSetFont`.

```
...
ft=XLoadFont(dpy,fuentes[cont/2]);
XFreeFontNames(fuentes);
XSetFont(dpy,gc,ft);

...
```

un posible resultado puede ser la figura 2.5.

Tenemos ya la fuente, que puede ser de ancho constante o proporcional, en el primer caso ya podemos usarla, pero en el segundo podemos requerir saber el tamaño de los distintos caracteres, todo ello se recoger en un estructura tipo `XFontStruct` que se actualiza con `XQueryFont`. Hay una llamada que realiza los dos pasos anteriores `XLoadQueryFont`. Cuando ya no necesitamos más una fuente podemos liberarla con `XFreeFontInfo`, pero antes realizamos `XUnloadFont`. Se pueden realizar de una vez con `XFreeFont`.

Cuando una fuente no es constante debemos atender a la métrica de la misma la cual nos define las medidas, en pixels, de una fuente como un todo e individualmente por caracter. El origen de todas ellas está definido por la línea base, `baseline`, que corresponde a un afila de pixels cercano a la mitad inferior de algunos caracteres. Todo esto se asemeja a estándares ya existentes (Adobe) Comentábamos que daremos información

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005



Figura 2.5: Otra fuente de texto

sobre la fuente como un todo, utilizando `XFontStruct` ya mencionada, y sobre caracteres `XCharStruct`. Ambas estructuras tienen nombres comunes pero que no representan lo mismo como veremos.

La estructura `XFontStruct` contiene información sobre la fuente como un todo.

```
typedef struct {
 XExtData *ext_data;
 Font fid;          /* identificador de la fuente */
 unsigned direction; /* indica hacia donde se escribe la fuente,
                        FontLeftToRight o FontRightToLeft */
 unsigned min_char_or_byte2;/* \'{\i}ndice al primer carÄcter*/
 unsigned max_char_or_byte2;/* al \'ultimo */
 unsigned min_byte1; /* 0 si son fuentes de un byte, en otro caso*/
 unsigned max_byte1; /* indican la primera y \'ultima fila que
                        existe en una fuente, hasta 256 como m\'aximo*/
 Bool all_chars_exist;  /* indica si todos los caracteres tienen
                        tama\~no >0*/
 unsigned default_char;/* car\'acter a tomar cuando se indica
                        uno inexistente */
 int n_properties; /* n\'umero de propiedades adicionales */
 XFontProp *properties; /* puntero para propiedades adicionales*/
 XCharStruct min_bounds;/* l\'{\i}mites m\'{\i}nimos para el universo
                        de caracteres de la fuente */
 XCharStruct max_bounds;/* l\'{\i}mites m\'aximos para el universo
                        de caracteres */
 XCharStruct *per_char; /* informaci\'on de cada carÄcter,
                        NULL si son proporcionales */
 int ascent;        /* extensi\'on desde la l\'{\i}nea base hasta el
                        l\'{\i}mites superior del mapa de pixels,
                        para espaciado */
 int descent;       /* extensi\'on desde la l\'{\i}nea base hasta el
                        l\'{\i}mites inferior del mapa de pixels */
} XFontStruct;
```

La estructura `XCharStruct` se describe a continuación. Hemos visto corresponde a varios campos de `XFontStruct` donde describen las medidas míinimas y máximas, así como para cada carácter.

```
typedef struct {
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
short lbearing; /* origen del lado izquierdo del caracter */
short rbearing; /* origen del lado derecho */
short width;    /* desplazamiento hasta el origen del
                        siguiente */
short ascent;   /* altura desde la l\'{\i}nea base al borde
                         superior del caracter */
short descent;  /* altura desde la l\'{\i}nea base al borde
                        inferior del caracter */
unsigned short attributes;/* sin uso est\'andar */
} XCharStruct;
```

Descritas las estructuras podemos comenzar a poner el texto donde queremos, con `XDrawString` o `XDrawString16` dadas una cadena, su longitud y el origen que deseamos para la línea base pintamos una cadena. Con llamadas como `XDrawImageString` o `XDrawImageString16` se dibuja en el rectángulo definido por `max_bounds` de manera análoga a como lo hace cualquier otra llamada para dibujar, es decir, la zona que lo rodea se pinta con el color del fondo. Finalmente con `XDrawText` o `XDrawText16` pintamos una o más cadenas siguiendo la estructura `XTextItem`.

```
typedef struct {
    char *chars;     /* puntero a la cadena */
    int nchars;      /* n\'umero de caracteres */
    int delta;       /* separaci\'on con la anterior */
    Font font;       /* Fuenta a usar */
  } XTextItem;
```

Con todo esto y conocidos los campos referentes al tamaño podemos jugar para posicionar donde nos apetezca. Existen otras llamadas útiles, si queremos conocer el ancho en pixels de una cadena utilizamos `XTextWidth` o `XTextWidth16`. Con `XTextExtents` o `XTextExtents16` obtenemos la cadena y la métrica de la fuente, evitando realizar repetidas llamadas a la estructura `XFontStruct`. La misma acción se realiza con `XQueryFontExtents` o `XQueryFontExtents16` pero consultando directamente al servidor sin pasar por la estructura.

Por defecto una fuente no tiene propiedades, pero si existen tendrán al menos las que se listan posteriormente, para cada implementación puede consultarse $< X11/Xatom.h >$. Con un átomo, `atom`, que le pasamos a `XGetFontProperty`.

```
#define XA_MIN_SPACE  /* Espaciado m\'{\i}nimo entre palabras */
#define XA_NORM_SPACE /* Espaciado normal entre palabras */
#define XA_MAX_SPACE  /* Espaciado m\'aximo entre palabras */
#define XA_END_SPACE  /* Espaciado adicional al final de las frases */
#define XA_SUPERSCRIPT_X /* Desplazamientos para super\'{\i}ndices */
#define XA_SUPERSCRIPT_Y
#define XA_SUBSCRIPT_X   /* Desplazamientos para sub\'{\i}ndices */
#define XA_SUBSCRIPT_Y
#define XA_UNDERLINE_POSITION  /* Desplazamiento del subrayado
a la l\'{\i}nea base */
#define XA_UNDERLINE_THICKNESS /* Ancho desde el subrayado a la
l\'{\i}nea base */
#define XA_STRIKEOUT_ASCENT /* Extensiones para l\'{\i}mites de carcter */
#define XA_STRIKEOUT_DESCENT
#define XA_ITALIC_ANGLE     /* \'angulo para it\'alica */
#define XA_X_HEIGHT      /* Habitualmente altura de x */
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
#define XA_QUAD_WIDTH      /* Ancho de m */
#define XA_WEIGHT       /* Grosor de la fuente */
#define XA_POINT_SIZE       /* Tama\~no del punto */
#define XA_RESOLUTION       /* Pixels por punto */
```

Si se desean añadir propiedades adicionales se utiliza la estructura `XFontProp`.

```
 typedef struct{
Atom name;
unsigned long card32;
 }XFontProp;
```

Comentar que puede modificarse la localización del directorio de las fuentes, con `XGetFontPath` lo leemos, con `XSetFontPath` se cambia, y con `XFreeFontPath` se libera.

2.5.4 Regiones

Se conoce como región a un conjunto arbitrario de pixels de la pantalla, habitualmente es una zona rectangular, varias o un polígono general. Dicha región puede especificarse con `XSetRegion`, estableciendo de esta forma el campo `clip_mask` de GC, resulta más cómodo que utilizando `XSetClipMask`, y es más flexible que `XSetClipRectangles`. El uso más inmediato es la localización de un evento `Expose` a una zona concreta, definida por el `clip_mask`.

Diversas acciones se pueden llevara cabo con las regiones, desde su creación con `XCreateRegion` o `XPolygonRegion`, la primera crea una región vacía a la que se le añaden rectángulos con `XUnionRectWithRegion` y otras funciones que realizan operaciones lógicas entre regiones como por ejemplo `XXorRegion` que nos da la diferencia entre la unión y la intersección, `XUnionRegion` que proporciona la unión, `XSubtractRegion` sustrae dos regiones y `XIntersectRegion` proporciona la intersección. Con la segunda, `XPolygonRegion`, creamos una región especificada con el formato similar al de la llamada `XDrawLines`, lo cual permite crear regiones no rectangulares. Y claro la opción de destruir con `XDestroyRegion`. Otras acciones como desplazar una región con `XOffsetRegion` o escalarla con `XShrinkregion` son posibles. También podemos comprobar la inclusión de un rectángulo en un a región con `XRectInRegion`, si es vacía con `XEmptyRegion`, si dos regiones son idénticas con `XEqualRegion`, si un punto está en una región con `XPointInRegion`.

2.5.5 Imágenes

Tenemos otras estructuras interesantes como son las imágenes. Las llamadas que manipulan imágenes utilizan una estructura que es como sigue

```
typedef struct _XImage {
 int width, height;    /* tama\~no */
 int xoffset;          /* desplazamiento en pixels en x */
 int format;           /* XYBitmap se representa por l\'{\i}neas de barrido pero cada lõ
 char *data;           /* puntero a los datos */
 int byte_order;       /* orden de los bytes de datos, LSBFirst, MSBFirst */
 int bitmap_unit;      /* cantidad por l\'{\i}nea de barrido 8, 16, 32 */
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 int bitmap_bit_order;/* LSBFirst, MSBFirst */
 int bitmap_pad;      /* 8, 16, 32 tanto para XY como ZPixmap */
 int depth;           /* profundidad */
 int bytes_per_line;  /* bytes por l\'{\i}nea */
 int bits_per_pixel;  /* bits por pixel (ZPixmap) */
 unsigned long red_mask;/* bits de cada pixel para cada plano de color, en ZPixmap */
 unsigned long green_mask;
 unsigned long blue_mask;
 char *obdata;        /* hook for object routines to hang on */
 struct funcs {       /* rutinas de manipulaci\'on */
  struct _XImage *(*create_image)();
  int (*destroy_image)();
  unsigned long (*get_pixel)();
  int (*put_pixel)();
  struct _XImage *(*sub_image)();
  int (*add_pixel)();
  } f;
} XImage;
```

Las imágenes podrán crearse `XCreateImage`, destruirse `XDestroyImage`, tomar la ventana o un mapa de pixels `XGetImage`, volcarse a una pantalla o mapa de pixels `XPutImage`, tomar el valor de un pixel `XGetPixel`, asignarlo `XPutPixel`, añadir un valor a todos los pixels con `XAddPixel`, crear una imagen de una zona de otra `XSubImage` y obtenerla de una zona de una ventana o mapa de pixels. Indicar que para crear mapas de pixels podremos partir de datos especificados en formato bitmap, tanto desde fichero `XReadBitmapFile`, crearlo `XWriteBitmapFile`, o ya desde código `XCreatePixmapFromBitmapData`. Algo sencillo que toma el mapa de la ventana con el boton central, y permite que dibujemos en la imagen puntos con `XPutPixel`, lo proporciona `image.c`, puede ser útil para redibujar de forma sencilla, pero no guardas la estructura del dibujo.

```
...
XImage    *imag;
...
 case ButtonPress:
   switch(ev.xbutton.button){
    case Button1:
     XPutPixel(imag,ev.xbutton.x,ev.xbutton.y,fore);
     XPutImage(dpy,win,gc,imag,0,0,0,0,thewin.width,thewin.height);
    break;
    case Button2:
     imag=XGetImage(dpy,win,0,0,thewin.width,thewin.height,AllPlanes,XYPixmap);
    break;
  }
 break;
...
```

2.5.6 Cursores

Los cursores también pueden manipularse, en definitiva son bitmaps, pero con particularidades como es su punto activo, que han facilitado la creación de rutinas específicas. Cada ventana puede tener un cursor asociado `XDefineCursor`, se puede desasignar `XUndefineCursor`. Si deseamos utilizar las formas habituales podemos recurrir

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

a $< X11/cursorfont.h >$, y cargarlo con `XCreateFontCursor`. Y finalmente deberemos liberarlo con `XFreeCursor`. Un ejemplo sencillo en `cursor.c`

```
...
Cursor    cursor;
...
cursor=XCreateFontCursor(dpy,XC_dot);
XDefineCursor(dpy,win2,cursor);
...
```

También podemos obtenerlo a partir del mapa de pixels `XCreatePixmapCursor` o `XCreateGlyphCur` o alterarlo con `XRecolorCursor`. Para consultar el hardware tenemos la comprobación del mejor tamaño con `XQueryCursor`. El ejemplo lo tenemos con `cursor_propio.c`, aunque quizás se introducen tipos de variables que veremos en la siguiente sección.

```
...
Cursor          cursor;
Pixmap          ima;
XColor          Fore,Back;
Colormap        cmap;
#define XC_circle_width 16
#define XC_circle_height 16
static char XC_circle_bits[] = {
 0x00, 0x00, 0x00, 0x00, 0xe0, 0x03, 0xf8, 0x0f, 0xfc, 0x1f, 0xfc, 0x1f,
 0xfe, 0x3f, 0xfe, 0x3f, 0xfe, 0x3f, 0xfe, 0x3f, 0xfe, 0x3f, 0xfc, 0x1f,
 0xfc, 0x1f, 0xf8, 0x0f, 0xe0, 0x03, 0x00, 0x00};
...
cmap=DefaultColormap(dpy,scr);
XAllocNamedColor(dpy,cmap,"white",&Fore,&Fore);
XAllocNamedColor(dpy,cmap,"black",&Back,&Back);

ima=XCreatePixmapFromBitmapData(dpy,DefaultRootWindow(dpy), XC_circle_bits, XC_
circle_width,XC_circle_height, fore,back,DefaultDepth(dpy,scr));
cursor=XCreatePixmapCursor(dpy,ima,None,Fore,Back,0,0);
XDefineCursor(dpy,win2,cursor);
...
```

2.6 Color

Una aplicación X normal, permite especificar el color del fondo y del borde de una ventana, los colores del cursor y los colores utilizados por el GC para pintar. Puede ser necesaria la incorporación de mayor número de colores, como puede ser la visualización de imágenes o la reproducción de imágenes realistas. A continuación veremos el proceso de traducción de valores de pixels a colores. Como el X es portable el tema del color es complicado para garantizar esa portabilidad.

Actualmente las pantallas trabajan principalmente con el modelo RGB para el color, se considera que cada pixel se obtiene combinando tres fósforos, el rojo, el verde y el azul. Combinándolos obtenemos toda la gama de colores, lo que para cada pixel requerirá la existencia de múltiples bits para codificar su color. Habitualmente se utiliza un mapa

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

de colores, `colormap`, para traducir el valor del pixel a sus correspondientes componentes RGB. El mapa de colores es lo que habitualmente se conoce como tabla *lookup*, donde el índice del pixel nos indica la posición de la celda a leer donde encontraremos los valores RGB. En una estación tendremos un rango dependiente del número de bits disponibles para especificar el RGB, pero el número de colores diferentes que pueden presentarse a la vez depende del número de planos de la pantalla. Esto último distingue las pantallas de mayor o menor eficiencia para color. El X es capaz de interactuar con distintas pantallas gracias a la estructura `visual`, la cual contiene información referente a como usar la pantalla en cada caso. Como ocurre con otras estructuras de X, es opaca y puede tomarse del padre, que si este ha tomado la de defecto puede igualmente tomarse con `DefaultVisual`, Dos campos de la estructura Visual son de interés, por un lado `map_entries` indica el número de celdas que el mapa de color actual creado con este Visual tiene. Por otro lado `class` indica la clase de Visual, que se corresponde a distintos tipos de monitor como son los monocromos `GrayScale` para lectura/escritura y `StaticGray` para solo lectura, cuando hay un índice para RGB tenemos si es lectura/escritura `PseudoColor`, y `StaticColor` para solo lectura, y como caso final, si hay varios índices (para cada fósforo) para el RGB en modo lectura/escritura `DirectColor` y `TrueColor` en solo lectura.

El *visual* no es solo un registro de las características del monitor, también expresa una forma de interpretar los recursos del monitor, es decir, para un monitor con un mapa de colores de lectura/escritura existen distintos visual. Notar que si el mapa es de solo lectura no se podrá modificar, aspecto interesante si se desea que el mapa sea compartido entre varios clientes, pero si no tenemos el color exacto que deseamos nos vemos limitados, cosa que conseguimos con los visuals más útiles, `PseudoColor`, y `DirectColor`. En los mapas de colores de lectura/escritura podemos tener celdas de lectura y de lectura/escritura, con estas podremos obtener el color deseado. Habitualmente es bastante utilizar el mapa de colores de defecto, que posee celdas de lectura/escritura que las rellenamos con nuestros requisitos.

Al crear una ventana se le asigna un visual que no puede ser modificado, la ventana raíz recibe el visual de defecto al inicializar el servidor. Recordando nuestros ejemplos iniciales, con `XCreateWindow` pasábamos a la ventana que creamos un visual por defecto, que es el que ha sido asignado a la ventana raíz. Este visual contiene lo que más usualmente se espera como cliente, para sistemas en color se suele asignar `PseudoColor`. Aunque nuestro servidor no pueda manejar varios mapas de colores por hardware será capaz de mantenerlos virtualmente. Un mapa de colores se crea con `XCreateColormap` aportando el dato de un visual, como vimos cada ventana puede mantener su propio mapa, pero solo si se requieren particularidades nos alejaremos del mapa de defecto. Los contenidos del mapa dependen del servidor.

Para aquellos mapas en los que sea posible leer y escribir un cliente podrá almacenar sus propios valores RGB en las celdas de color. De todas formas si el número de celdas es escaso se aconseja en lugar de exigir colores exactos tomar uno de los predefinidos. Cuando un cliente intenta alojar una celda de solo lectura, no especifica el valor del pixel sino que pide un color, y se le devuelve el valor del pixel que apunte al color disponible más cercano. Este valor es utilizable para cualquiera de los atributos de ventanas o GC que hemos visto hasta ahora. Con `XAllocColor` se devuelve el valor de pixel más con

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

RGB más cercano al solicitado, para hacerlo por nombre utilizamos `XAllocNamedColor`. Para obtener el valor RGB asociado a un nombre utilizamos `XParseColor`. Las llamadas mencionadas hacen uso de la estructura `XColor`.

```
typedef struct {
 unsigned long pixel;    /* valor del pixel */
 unsigned short red, green, blue;     /* valores RGB de 16 bits */
 char flags;             /* DoRed, DoGreen, DoBlue */
 char pad;
} XColor;
```

Cuando se devuelve un color, no ocurrirá en la mayoría de los casos la coincidencia entre el deseado y el obtenido, por ello el cliente debería chequear el RGB devuelto.

Para simplificar la especificación de los colores y promover su compartición existe un fichero ASCII que contiene una base de colores accesible en `/usr/lib/X11/rgb.txt`.

Es posible también especificar los colores utilizando valores hexadecimales

```
#RGB
#RRGGBB
#RRRGGGBBB
#RRRRGGGGBBBB
```

En el siguiente ejemplo alojamos pedimos colores al mapa de colores, tanto por valores RGB como por nombre. Por comprobarlo, a continuación consultamos los valores RGB asociados a un valor de pixel, es decir, a un índice de la tabla. Tomado de `pido_color.c`.

```
 ...
 XColor color,color2;
 ...
 color.red=col-1;
 color.green=0;
 color.blue=0;
 if (!XAllocColor(dpy,cmap,&color))
  printf("Error\n");
 XQueryColor(dpy,cmap,&color);
 XAllocNamedColor(dpy,cmap,"red",&color,&color2);
 XQueryColor(dpy,cmap,&color);
 ...
```

Cuando el cliente usa celdas de solo lectura es más lógico llamar a `XParseColor`, aunque pueden aportarse los valores RGB directamente, que puede realizarse con enteros, con hexadecimal, desplazando ...

Si nos encontramos en un sistema con mapa de color de lectura/escritura y un cliente necesita colores exactos podremos alojar celdas y almacenar entonces colores en ellas. Para cambiar valores RGB podemos utilizar `XAllocColorCells`, posteriormente se establecen los colores de las celdas con `XStoreColor`, `XStoreColors` o `XStoreNamedColor` Por ejemplo en `aloja_color.c` se colocan en el mapa cinco colores privados.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
...
#define   COLOR  5
XColor    colores[COLOR];
int       plane_mask[1];
int       colors[COLOR];
int       ncolors=COLOR;
char      nombre[5][15];

strcpy(nombre[0],"violet");
strcpy(nombre[1],"salmon");
strcpy(nombre[2],"coral");
strcpy(nombre[3],"aquamarine");
strcpy(nombre[4],"darkorchid");

...
if (!XAllocColorCells(dpy,cmap,False,plane_mask,0,colors,ncolors))
 printf("Error\n");
for (i=0;i<5;i++){
 if (!XParseColor(dpy,cmap,nombre[i],colores[i]))
   printf("Error Parse\n");
   colores[i].flags=DoRed | DoGreen | DoBlue;
   colores[i].pixel=colors[i];
 }
XStoreColors(dpy,cmap,colores,ncolors);
...
```

Por otro lado `XAllocColorPlanes` se utiliza cuando quieres ser capaz de variar una componente de color primario de algo ya dibujado sin tener que redibujarlo todo.

También es posible superponer sin destruir un conjunto de gráficos sobre otro, la capa anterior no será visible, pero con un simple refresco sobre los planos podremos alterar este hecho. Para esto se utiliza el truco de dibujar sobre algún plano sin modificar el resto.

Llegados a este punto podemos abordar los mapas de colores. Un mapa por hardware dispone de registros desde los que toma los valores RGB para pintar. Normalmente los equipos cuentan con solo uno de estos mapas, lo que quiere decir que todas las ventanas se interpretan con dicho mapa. Si el mapa no puede modificarse se denomina inmutable, pero afortunadamente los sistemas suelen permitir que se escriban nuevos valores en el mapa de color por hardware, se denomina en este caso como modificable. En este caso se pueden mantener varios mapas de colores y copiarse en el momento que se vaya a utilizar, estos son los mapas virtuales. Si un cliente no puede obtener las celdas que necesita del mapa ya instalado, puede crear uno propio. Como es un recurso compartido ocurre que si un cliente copia un nuevo mapa, todos los clientes que estaban usando el mapa previo se verán con falso color. Es un problema pensar que el administrador de ventanas se encarga de instalar y desinstalar los mapas en función de la ventana, la mayoría no lo hace, por lo que el cliente se debe ocupar de ello. La creación se realiza con `XCreateColormap`, lo cual permite crear un mapa virtual sin entredas, en el que podremos colocar celdas a nuestro gusto, o totalmente modificable con lo que el mapa será totalmente privado. Se elimina con `XFreeColormap`, se instala con `XInstallColormap`, y se realiza la acción contraria con `XUninstallColormap`. Podemos obteber los instalados `XListInstalledColormap`. Llevar las entradas desde un a otro y liberar las anteriores `XCopyColormapAndFree`. Podemos crear un mapa estándar con `XSetStandardColormap`, recuperar información acerca de un mapa estándar con `XGetStandardColormap` o asignar un mapa como atributo de

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

una ventana XSetWindowColormap. Para referirnos a los mapas estándares utilizamos átomos de forma similar a las fuentes.

- XA_RGB_DEFAULT_MAP define parte del mapa de defecto, para 8 planos da 216 colores distribuidos y deja 40 para otros programas.
- XA_RGB_BEST_MAP proporciona el mejor mapa RGB disponible en la pantalla.
- XA_RGB_RED_MAP, XA_RGB_GREEN_MAP, XA_RGB_BLUE_MAP definen mapas totalmente del color primario al que correspondan.
- XA_RGB_GRAY_MAP proporciona el mejor mapa de grises para la pantalla.

Una aplicación utilizando un mapa estándar obtiene una estructura al leerlo del servidor

```
typedef struct {
 Colormap colormap;   /* identificador del mapa de color */
 unsigned long red_max;  /* los max dan el m\'aximo del tono */
 unsigned long red_mult; /* los mult dan un factor para escalar */
 unsigned long green_max;  /* en funci\'on de los bits usados */
 unsigned long green_mult;
 unsigned long blue_max;
 unsigned long blue_mult;
 unsigned long base_pixel; /* valor base que se a\~nade a cada pixel */
 VisualID visualid;
 XID killid;
} XStandardColormap;
```

En colormap.c instalamos uno de los mapas de color por defecto.

```
#include <X11/Atom.h>

...

XSetWindowAttributes   atributos;
Colormap       cmap;
XStandardColormap      info;

...

XGetStandardColormap(dpy,DefaultRootWindow(dpy),&info,XA_RGB_DEFAULT_MAP);
cmap=info.colormap;
atributos.background_pixel=back;
atributos.border_pixel=fore;
atributos.colormap=cmap;

win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
DefaultDepth(dpy,scr),InputOutput,DefaultVisual(dpy,scr),
CWColormap | CWBackPixel | CWBorderPixel, &atributos);

...
```

Pero es posible personalizar el mapa de colores, para utilizar mapas de colores donde todos los colores primarios no tengan la misma resolución, etc. Con esto tenemos libertad para crearlos a nuestra necesidad, siempre claro está dentro de los límites del hardware.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Como X soporta muchos tipos de sistemas, aún cuando el manejo del color es sencillo, se debe tener cuidado para segurar que la aplicación correrá tanto en blanco y negro como color. Excepto si nuestra aplicación tiene necesidades particulares, lo lógico es tomar el visual de defecto y modificar los colores en función de nuestra aplicación. Si la aplicación tiene sus necesidades deberemos elegir el mejor visual entre los posibles, con llamadas como `XMatchVisualInfo` y `XGetVisualInfo` se utilizan para recoger dicha información. En `profundidad-visual.c` se leen las profundidades disponibles para una pantalla, y para cada una de ella los visuals correspondientes.

```
...
int            *prof,i,j,cont,cont2;
XVisualInfo *listavisual,template;
...
prof=XListDepths(dpy,scr,&cont);
for (i=0;i<cont;i++){
 printf("Profundidad %d \n",prof[i]);
 template.screen=scr;
 template.depth=prof[i];
 listavisual=XGetVisualInfo(dpy, VisualDepthMask | VisualScreenMask, &templa
 te,&cont2);
 printf("Visuals %d\n",cont2);
  for (j=0;j<cont2;j++){
   switch(listavisual[j].class){
     case PseudoColor:
      printf("PseudoColor \n");
     break;
     case StaticColor:
      printf("StaticColor \n");
     break;
     case DirectColor:
      printf("DirectColor \n");
     break;
     case TrueColor:
      printf("TrueColor \n");
     break;
     case GrayScale:
      printf("GrayScale \n");
     break;
     case StaticGray:
      printf("StaticGray \n");
     break;
   }
  }
}
...
```

De manera totalmente análoga al mapa de colores, podremos asignar un visual determinado al crear una ventana, simplemente indicándolo en `XCreateWindow` sustituyendo al tradicional hasta ahora `DefaultVisual(dpy,scr)`. Al recuperar información sobre estas estructuras obtenemos:

```
typedef struct {
     Visual *visual;           /* puntero a la estructura interna */
     VisualID visualid;
     int screen;
     unsigned int depth;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
    int class;  /* clase del visual */
    unsigned long red_mask;  /* utilizado con DirectColor y TrueColor */
    unsigned long green_mask;
    unsigned long blue_mask;
    int colormap_size;/* indica el n\'umero de valores de pixels para este visual */
    int bits_per_rgb; /* cuantos bits de cada valor se utilizan para en ca\~n\'on */
  } XVisualInfo;
```

2.7 Eventos

En X un evento va a informarnos de una interacción con un dispositivo, o será generado por alguna rutina de Xlib. Un cliente en X comparte su existencia con otros varios, no puede limitarse a esperar por una entrada, debe estar preparado para los posibles efectos que otros clientes le puedan causar, como por ejemplo el que otro cliente tape a nuestra aplicación. Los eventos se producen de forma asíncrona y se encolan en la cola del cliente que espera recibirlo. habitualmente los clientes lo tratan uno a uno antes de pasar al siguiente. Ya hemos comentado que al programar el manejo de eventos primero debemos identificar los eventos a tratar, luego se visualiza la ventana, y finalmente se entra en un bucle de manejo de eventos. La selección de eventos en nuestro primer ejemplo está realizada con `XSelectInput`, con dicha llamada se pueden expresar por combinación (or), los eventos que se procesarán con este cliente.

Un evento se implementa como información almacenada en una estructura, todos los eventos tienen una estructura asociada, cada una de ellas, unas 30, se describe en $< X11/Xlib.h >$, pero poseen los siguientes campos comunes

```
typedef struct {
      int type;   /* tipo de evento */
      unsigned long serial;/* describe la \'ultima procesada para depurar */
      Bool send_event;/* true si se debe a XSenEvent */
      Display *display;/* pantalla, servidor, donde se recogi\'o el evento */
      Window window;   /* ventana que recibe el evento */
    } XAnyEvent;
```

La unión `XEvent` contiene todas las estructuras, un cliente determina el tipo por medio del campo `type` y con ello la estructura adecuada.

Una vez que un evento ocurre, su estructura se coloca en la cola de eventos, en función del momento en que ocurrió. Cada cliente dispondrá de su cola propia, en la que se almacenan los eventos seleccionados previamente.

El tratamiento realizado en el bucle de eventos puede verse en cualquier programa X. Determinar el tipo del evento entrante es sencillo, en el fichero $< X11/X.h >$ pueden verse el listado de tipos de eventos.

Para recoger los eventos de la cola existen variadas rutinas.

- `XGetEvent` toma el siguiente evento de cualquier tipo sobre cualquier ventana de la cola.
- `XMaskEvent` captura el evento que coincida con una máscara de cualquier ventana, espera si no lo hay.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

- `XCheckMaskEvent` como el anterior pero devuelve `False` si no existe ningún evento de ese tipo.

- `XWindowEvent` captura el próximo evento de cierto tipo en cierta ventana, espera si no lo hay.

- `XCheckWindowEvent` como el anterior pero devuelve `False` si no encuentra uno válido.

- `IfEvent` busca un evento de cierto tipo y espera pr él si no lo encuentra en la cola.

- `XCheckIfEvent` como el anterior, pero no espera, devuelve `False`.

- `XPeekEvent` toma el siguiente evento de cualquier tipo y para cualquier ventana de la cola, sin sacarlo de ella.

- `XPeekIfEvent` espera por un evento de cierto tipo, vaciando el resto de la cola.

- `CheckTypedEvent` busca en la cola el evento *más viejo* de un tipo determinado.

- `CheckTypedWindowEvent` busca en la cola el evento más viejo de un tipo determinado para una ventana determinada.

- `EventsQueued` retorna el número de eventos pendientes. Hay varios modos.

- `Pending` devuelve el número de eventos pendientes. tipo determinado.

- `PutBackEvent` coloca un evento de nuevo en la cola, será el próximo a recibir.

- `GetMotionEvent` captura todos los eventos de movimientos para una ventana en un intervalo de tiempo.

Ciertas rutinas (if) permiten especificar un predicado que devuelve verdadero o falso en base a las características del evento.

Como se comentaba previamente, un cliente selecciona los eventos que tiene en cuenta una ventana por medio de `XSelectInput`, llamada a la que le pasa las máscaras que identifican los eventos combinadas con el operador OR, a continuación indicamos las máscaras posibles

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Máscara de evento	circunstancias
NoEventMask.	Ningún evento
KeyPressMask	Se ha pulsado una tecla, un código identifica a la misma, como es dependiente del servidor usaremos `XLookupString` para traducirlo a un código portable como es el `keysym`
KeyReleaseMask	Misma situación que el anterior, pero para el caso de soltar una tecla
ButtonPressMask	Activado al pulsar un botón del puntero
ButtonReleaseMask	Como anterior pero soltando
EnterWindowMask	Entrada de puntero en una ventana
LeaveWindowMask	Salida del puntero de una ventana
PointerMotionMask	Selecciona todos los eventos de movimiento del puntero
PointerMotionHintMask	Reduce el número de eventos generados, normalmente se obliga a enviar un solo evento cuando el puntero se mueve hasta que cambie el estado de una tecla o botón
Button1MotionMask	Movimiento con botón 1 pulsado
Button2MotionMask	Movimiento con botón 2 pulsado
Button3MotionMask	Movimiento con botón 3 pulsado
Button4MotionMask	Movimiento con botón 4 pulsado
Button5MotionMask	Movimiento con botón 5 pulsado
ButtonMotionMask	Movimiento con algún botón pulsado
KeymapStateMask	Cambio de estado del teclado
ExposureMask	Cualquier exposición
VisibilityChangeMask	Cambio en visibilidad
StructureNotifyMask	Cambio en la config. de una ventana
ResizeRedirectMask	Cambio de tamaño
SubstructureNotifyMask	Notificación reconfig. de los hijos
SubstructureRedirectMask	Reconfig. directa de los hijos
FocusChangeMask	Utilizado para controlar donde el puntero está activo en un instante
PropertyChangeMask	Cambio en la propiedad de una ventana
ColormapChangeMask	Cambio en el mapa de color
OwnerGrabButtonMask	Modifica el manejo de los eventos de puntero

La correspondencia entre la máscara de evento, el tipo y la estructura es como sigue

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Máscara de evento	Tipo de evento	Estructura
KeyPressMask	KeyPress	XKeyPressedEvent
KeyReleaseMask	KeyRelease	XKeyReleasedEvent
ButtonPressMask	ButtonPress	XButtonPressedEvent
ButtonReleaseMask	ButtonRelease	XButtonReleasedEvent
OwnerGrabButtonMask		
KeymapStateMask	KeymapNotify	XKeymapEvent
PointerMotionMask	MotionNotify	XPointerMovedEvent
PointerMotionHintMask		
ButtonMotionMask		
Button1MotionMask		
Button2MotionMask		
Button3MotionMask		
Button5MotionMask		
EnterWindowMask	EnterNotify	XEnterWindowEvent
LeaveWindowMask	LeaveNotify	XLeaveWindowEvent
FocusChangeMask	FocusIn	XFocusInEvent
	FocusOut	XFocusOutEvent
ExposureMask	Expose	XExposeEvent
Seleccionado por		
`graphics_expose` en el GC	GraphicsExpose	XGraphicsExposeEvent
	NoExpose	XNoExposeEvent
ColormapChangeMask	ColormapNotify	XColormapEvent
PropertyChangeMask	PropertyNotify	XPropertyEvent
VisibilityChangeMask	VisibilityNotify	XVisibilityEvent
ResizeRedirectMask	ResizeRequest	XResizeRequestEvent
StructureNotifyMask	CirculateNotify	XCirculateEvent
	ConfigureNotify	XConfigureEvent
	DestroyNotify	XDestroyEvent
	GravityNotify	XGravityEvent
	MapNotify	XMapEvent
	ReparentNotify	XReparentEvent
	UnmapNotify	XUnmapEvent
SubstructureNotifyMask	CirculateNotify	XCirculateEvent
	ConfigureNotify	XConfigureEvent
	DestroyNotify	XDestroyEvent
	GravityNotify	XGravityEvent
	MapNotify	XMapEvent
	ReparentNotify	XReparentEvent
	UnmapNotify	XUnmapEvent
SubstructureRedirectMask	CirculateRequest	XCirculateRequestEvent
	ConfigureRequest	XConfigureRequestEvent
	MapNotify	XMapRequestEvent
siempre seleccionado	MappingNotify	XMappingEvent
siempre seleccionado	ClientMessage	XClientMessageEvent
siempre seleccionado	SelectionClear	XSelectionClearEvent
siempre seleccionado	SelectionNotify	XSelectionEvent
siempre seleccionado	SelectionRequest	X SelectionRequestEvent

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

En las estructuras de eventos se incluye el identificador de una ventana que es la denominada ventana de evento, la cual no siempre coincide con la ventana en la que se produjo el mismo, todo depende de la jerarquía de ventanas y las posibles alteraciones mediante el campo do_not_propagate_mask, ya mencionado, de los atributos de la ventana. La ventana fuente es la de menor tamaño que contiene al puntero en el momento de producirse el evento, si esta lo tiene seleccionado lo trata, en otro caso se comprueba que no esté seleccionado como de no propagación y si no es así lo envía a sus ancestros.

2.8 Teclado y Puntero

2.8.1 Teclado

El teclado debe ser portable igual que el color entre un sistema y otro, cuando se produce un evento de teclado se trabaja con la estructura XKeyEvent

```
typedef struct {
      int type;    /* tipo de evento */
      unsigned long serial;/* describe la \'ultima procesada para depurar */
      Bool send_event;/* true si se debe a XSenEvent */
      Display *display;/* pantalla, servidor, donde se recogi\'o el evento */
      Window window;    /* ventana que recibe el evento */
      Window root;      /* ventana ra\'{\i}z donde ocurri\'o */
      Window subwindow; /* ventana hija */
      Time time; /* milisegundos */
      int x,y; /* coordenadas en la ventana del evento */
      int x_root,y_root; /* coordenadas relativas a la ra\'{\i}z */
      unsigned int state; /* mDscara de la teclka o bot\'on */
      unsigned int keycode; /* tecla */
      Bool same_screen; /* flag de monitor coincidente */
    } XKeyEvent;
```

El campo keycode nos identifica la tecla con un número entre 7 y 255 para ese servidor, como se indicó con anterioridad para permitir la portabilidad se traduce a un código común, el keysym utilizando XLookupString. Los códigos en dicho formato pueden consultarse en $< X11/keysymdef.h >$. La plicación debe ocuparse posteriormente de tratar cada caracter. Vemos el ejemplo, teclas.c, para salir de la aplicación si se pulsa Q (mayúscula).

```
#include <X11/keysym.h>
...
int             caracter,buf=20;
char            buffer[20];
XComposeStatus  compo;
KeySym          key;
...
  case KeyPress:
    caracter=XLookupString(&ev,buffer,buf,&key,&compo);
    if(key==XK_Q){
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
      XFreeGC(dpy,gc);
      XDestroyWindow(dpy,win);
      XCloseDisplay(dpy);
      exit(1);
    }
  break;
```

Una vez que disponemos del `keysym` podemos traducir de forma local a cadena, en realidad esta traducción para las teclas sencillas (no de control) ya se vuelca en `buffer`, y `caracter` devuelve además el número de caracteres que contiene. Simplemente añadiendo una líneas al anterior ejemplo podemos escribir la cadena que interactivamente introducimos.

```
#include <string.h>
...
      caracter=XLookupString(&ev,buffer,buf,&key,&compo);
      strcat(string,buffer);
      printf("Cadena %s\n",string);
      if(key==XK_Q){
...
```

También puede alterarse el significado de las teclas de control, para ello utilizamos `XRebindKeysym`. Hay eventos asociados a dichas teclas, e incluso nosotros podemos establecer una de dichas teclas (hay 4 libres). Se utiliza `XGetModifierMapping`, `XInsertModifiermapEntry` y `XDeleteModifiermapEntry`.

Finalmente indicar que es posible modificar el mapeo de código de teclado a keysym con `XChangeKeyboardMapping`. El actual puede recuperarse con `XGetKeyboardMapping`.

2.8.2 El puntero

El puntero produce eventos al moverse, cuando cruza los bordes, cuando pulsa botones, todo ello puede ser de interés para nuestra aplicación.

Si pensamos en primer lugar en el evento de movimiento del puntero, es claro que una opción es seguir todos los movimientos que este haga, `PointerMotionMask`, pero además tenemos la posibilidad de recoger solo la posición tras haber realizado el movimiento `PointerMotionHintMask`. En ambos casos cuando se produce el evento puede recurrirse a `XQueryPointer` para recoger la posición del puntero, o tomar ese valor de la estructura del evento directamente. Otra opción, si el servidor dispone de un *buffer*, es leer el camino seguido por el puntero.

Y por otro lado tenemos los eventos asociados a los botones, que ciertamente pueden combinarse en su tratamiento con los anteriores. Ya son conocidos por ejemplos previos, pero no está de más repetir la existencia de los eventos para pulsar y soltar un botón del ratón, respectivamente `ButtonPress` y `ButtonRelease`. Indicar que el evento de soltar se asocia siempre a la ventana en la que se pulsó (aunque se produzca fuera), a menos que se altere con `OwnerGrabButton` y `XChangeActivePointerGrab`.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Ciertas aplicaciones pueden querer modificar el mapeo entre el puntero físico y los botones, por ejemplo para un zurdo. Con `XGetPointerMapping` se lee el actual, se establece con `XSetPointerMapping`, y se actualiza con `XRefreshPointerMapping`. También es posible llevar el puntero a una posición por programa con `XWarpPointer`.

También interesa en ocasiones el efecto de abandonar o entrar en una ventana, `LeaveNotify` y `EnterNotify` respectivamente.

Por cierto se ofrecen llamadas para configurar las denominadas preferencias del teclado y del ratón, es decir, determinar el tiempo para el *beep*, para repetir tecla, etc. Si se quieren indicar todas de golpe se utiliza `XGetKeyboardControl` para leerlas, y `XChangeKeyboardControl` para establecerlas, con todas sus peculiaridades. Para el ratón se utilizan `XGetPointerControl` y `XSPointerControl`.

2.9 Manejando recursos

La geometría de una aplicación en X Window puede especificarse como hemos visto hasta ahora por programa, pero si observamos los ejemplos también aparecen `argc` y `argv` en las llamadas al crear, luego se tienen en cuenta las indicaciones de línea de comando. Existe otra posibilidad análoga a la indicada en el capítulo previo para los clientes X, etiqueta que ya tiene nuestra aplicación, consistente en trabajar con un fichero de `.Xdefaults` que se encuentra en nuestro directorio inicial (*home*). Si este fichero no existe los valores por defecto se toman del directorio `/usr/lib/X11/app-defaults` o por medio de la variable de entorno `XENVIRONMENT`, buscando por el nombre del cliente.

En dichos ficheros de configuración las líneas tienen una estructura del modo `objeto...subobjeto...atributo: valor`, siendo posible especificar `*` para referirnos a cualquier elemento. Según esto podemos tener un fichero de la forma

```
...
hola.foreground    120
hola.background    0
*font              -adobe-helvetica-bold-r-normal--*-120-*-*-*-*-iso8859-*
...
```

Nos falta decir que el nombre de un objeto, una ventana, coincide con el del cliente, la aplicación. Si indicamos un atributo con `*` al principio estaremos marcando el estilo de cualquier cliente. Todo sin necesidad de compilar de nuevo.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Capítulo 3

Programación en Motif

3.1 Introducción

Motif es una biblioteca de rutinas que hace más cómodo y fácil la creación de interfaces de usuario en un entorno X. La biblioteca Motif esconde al usuario la mayoría de las tareas de manejo de eventos de bajo nivel (nivel Xlib), por lo que se pueden crear interfaces de usuario bastante sofisticadas sin tener que entrar en toda la complejidad del X. El Motif se basa en la definición de los widgets. Un *widget* es simplemente un elemento de una interface de usuario - un botón, una barra deslizante, un área de texto, etc. - que posee una serie de recursos configurables por el usuario. Aunque la programación en Motif es ligeramente más sencilla que en X, aún encierra cierta dificultad. Por tanto la mejor forma de introducirse en Motif es a partir de un sencillo programa; y a partir del mismo ir viendo las posibilidades que ofrece Motif a la hora de definir una interface de usuario. A continuación se pasa a mostrar el primer programa en Motif, que muestra una ventana con un texto.

```
#include <Xm/Xm.h>
#include <Xm/Label.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,label;

void      main(int, char **);

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;

  toplevel=XtAppInitialize(&app_context,"hola",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Hola Mundo",
           char_set)); n++;
  label=XmCreateLabel(toplevel,"etiqueta",args,n);

  XtManageChild(label);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005



Figura 3.1: Progama "Hola Mundo"

```
}
```

Para compilar el anterior ejemplo se debe introducir en la línea de comandos:

```
cc -o hola-motif hola-motif.c -I/usr/X11/include -L/usr/X11/lib -lXm -lXt
-lX11 -Xext -lXpm
```

suponiendo que las bibliotecas y los ficheros de definición se encuentran en los directorios `/usr/X11/lib` y `/usr/X11/include` respectivamente. En caso de no encontrarse en los anteriores directorios, se debe modificar con los directorios correctos. Un aspecto importante en muchos sistemas es el orden de las bibliotecas en la línea de comandos. En muchos casos alterar dicho orden puede producir mensajes del compilador de referencias no encontradas o lo que puede ser peor, compilar el programa pero dar errores de ejecución bastante difíciles de depurar. En la figura 3.1 se puede ver la ventana que se obtiene al ejecutar el programa.

A continuación se describen los elementos principales del programa anterior. Para ello, las líneas más importantes se numerarán para poder referenciarlas en la explicación del programa.

```
#include <Xm/Xm.h>
#include <Xm/Label.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,label;

void      main(int, char **);

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;

1: toplevel=XtAppInitialize(&app_context,"hola",NULL,0,&argc,argv,NULL,NULL,0);
2: n=0;
3: XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Hola Mundo",
            char_set)); n++;
4: label=XmCreateLabel(toplevel,"etiqueta",args,n);
5: XtManageChild(label);
6: XtRealizeWidget(toplevel);
7: XtAppMainLoop(app_context);
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

La primera línea y más importante, ya que inicializa todo los relacionado con Xlib y Motif y da como resultado la ventana principal de la aplicación:

```
1: toplevel=XtAppInitialize(&app_context,"hola",NULL,0,&argc,argv,NULL,NULL,0);
```

La rutina `XtAppInitialize` tiene varios parámetros. El primero es un puntero a la variable que contendrá el contexto de la aplicación. La segunda es el nombre de la clase de la aplicación y se utilizará para configurar los recursos de los distintos widgets de la aplicación. El tercer y cuarto parámetro tienen que ver con el orden en que los parámetros pasados en la línea de comandos son buscados. En casos excepcionales se modificarán y lo normal es que estén como en el ejemplo. El quinto y sexto parámetro son `argc` y `argv` del main. Estos parámetros se pasan porque la rutina analiza la línea de comandos para buscar opciones relativas al entorno X, interpretándolas y eliminándolas. Por eso es necesario pasar el puntero a `argc`, ya que en la rutina se puede modificar su valor. Aunque exista la seguridad de que en la línea de comando no existe ningún parámetro relacionado con X, no se puede poner NULL y 0 en estos parámetros. El siguiente parámetro esta relacionado con la búsqueda del fichero de recursos y los dos últimos son la lista de recursos utilizados en la creación de la ventana principal y el número de elementos de dicha lista. El uso de las listas de recursos se verá en el siguiente apartado. En las líneas segunda y tercera

```
2: n=0;
3: XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Hola Mundo",
            char_set)); n++;
```

se asigna un valor al recurso `XmNlabelString`. Este recurso es el que va a contener el mensaje que aparece cuando la aplicación se ejecuta. El resto de los recursos van a tener los valores por defecto. La creación y manejo del widget se va a realizar en las líneas cuarta y quinta.

```
4: label=XmCreateLabel(toplevel,"label",args,n);
5: XtManageChild(label);
```

Hay que indicar que el acto de manejar un widget es lo que lo hace visible y no la creación del mismo. En la línea que crea el widget se indican varios parámetros. El primero es el widget padre del widget creado. Todos los widgets van a tener un padre, excepto el widget principal de la aplicación (en el ejemplo `toplevel`). A continuación se indica el nombre del widget. Este nombre se utiliza para poder asignar un valor a los recursos y también para indicar en caso de error, cuál ha sido el widget que ha fallado. Por último, se tiene el vector de argumentos y el número de elementos que contiene. Este vector como ya se ha indicado contiene el valor de los recursos a los que se les ha dado un valor en la creación del widget. Una vez que se ha creado, la rutina devuelve un valor que se utilizará para referenciar al widget en el resto del programa. Un aspecto muy importante es el que afecta a los ficheros de definición (`includes`) en programación en Motif. Algunos compiladores producen programas de funcionamiento errático si falta alguno de estos ficheros. Por tanto es conveniente que para cada widget que se utilice, se incluya el fichero de definiciones correspondiente (en el ejemplo `Xm/Label.h`). En la línea

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
6: XtRealizeWidget(toplevel);
```

se activa el widget toplevel. El hecho de activar un widget implica la creación de la ventana de la aplicación en la pantalla, y el emplazamiento de todos los widgets dentro de la misma. Normalmente sólo es necesario activar el widget principal de la aplicación, es decir, el que se crea con `XtAppInitialize`. Por último en la línea 7 es donde empieza el bucle de eventos.

```
7: XtAppMainLoop(app_context);
```

A esta rutina se le pasa como argumento el contexto de la aplicación que se obtiene cuando se inicializa la aplicación con la rutina `XtAppInitialize`. A partir de aquí el programa solo responderá a eventos que recibe del usuario y responderá a los mismos mediante los *Callbacks*, que son procedimientos que se activan cuando un evento se produce en un determinado widget.

Y esto es lo que es un programa en Motif. Resumiendo, el programa crea un widget shell denominado toplevel y un widget del tipo label que contiene la frase "Hola Mundo" como hijo suyo. Realiza el widget shell y a continuación empieza el bucle de eventos. Hacer notar que todo programa en Motif, por muy complicado que parezca hace básicamente lo visto en este pequeño programa de ejemplo.

3.2 Configuración de los widgets

Los widgets tienen un gran número de recursos tales como el color del fondo, tamaño de texto, tipo de fuente, etc. que poseen unos valores por defecto y que se pueden modificar por el usuario o el programador de la aplicación. La configuración de los widgets por parte del usuario es fundamental en aplicaciones gráficas, ya que el entorno de desarrollo no siempre coincide con el de aplicación. Así elementos como el número de colores disponibles y tamaño de la pantalla, influyen de forma importante en la apariencia final de una aplicación. Por otro lado, se encuentran los gustos personales del usuario que no van coincidir con los del diseñador, por lo que los temas relacionados con el aspecto se deben poder definir por el usuario y no quedar exclusivamente en manos del programador.

La flexibilidad de configuración de una aplicación debe tener un límite, ya que la cantidad de control dada al usuario nunca será tal que puede alterar el funcionamiento de la aplicación. Esto se puede ver por ejemplo si a un botón de "Borrar Fichero" se le puede cambiar el texto y poner "Guardar Fichero", lo que produciría un resultado erróneo de la aplicación.

Cada uno de los widgets disponibles se encuentra documentado en el *Manual del Programador de Motif*. Además la misma descripción se encuentra como páginas de manual en cualquier sistema donde esté instalado el *OSF/Motif*. Aparte de la descripción del widget, existe una lista de los recursos del widget. Algunos de los recursos serán propios y otros son heredados de otra superclase a la que pertenezca. Para cada widget se tienen las siguientes entradas.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Name	Class	Type	Default	Access
XmNactivateCallback	XmCCallback	XtCallbackList	NULL	C
XmNarrowDirection	XmCArrowDirection	unsigned char	Xm_ARROW_UP	CSG

El primero, *Name*, da el nombre del recurso. El segundo, *Class*, es el nombre de la clase a la que pertenece el recurso. La entrada, *Type*, es bastante importante ya que da el tipo del recurso. En muchos casos es necesario prestar bastante atención ya que existen tipos de nombres muy similares pero que no se pueden intercambiar. Por ejemplo existen los tipos `String` y `XmString` que aunque puedan parecer iguales, no los son. Otro posible error es asumir que tipos como `Position` o `Dimension` puedan ser enteros. La cuarta columna es el valor por defecto que tiene el recurso. Por ultimo, la quinta columna indica el tipo de acceso que tiene el recurso. El tipo acceso es una combinación de las letras *C*, *S* y *G*; que vienen de Creation, Set y Get. Así *C*, indica que el valor solo puede ser establecido cuando se crea el widget. *S* indica que puede ser modificado en cualquier momento y *G* que el valor puede ser leído también en cualquier momento.

3.2.1 Configuración por parte del usuario

Como se vió en el capítulo 1, en los ficheros `/usr/lib/X11/app-defaults/<class-name>` y `.Xdefaults`, se puede establecer la configuración de las aplicaciones X. Para poder definir un determinado recurso en alguno de los ficheros anteriores, es necesario identificar la aplicación y dentro de la misma el widget. Para identificar la aplicación se utiliza el nombre que se dió a la misma en la rutina de inicialización `XtAppInitialize`. En el programa ejemplo se dio el nombre `hola`. Para identificar el widget también se utiliza el nombre que se le dio en la creación. En el ejemplo, al widget label se le dio el nombre `etiqueta`. Una vez identificada la aplicación y el widget, se puede proceder a modificar el valor de un recurso. Esto se realiza poniendo el nombre de la aplicación, seguido del widget y por último el nombre del recurso sin el prefijo XmN o XmC que aparece en la página de manual. Cada uno de los nombres separado por un punto y después del nombre del widget se colocan dos puntos y a continuación el valor asignado al widget. Por ejemplo para cambiar el ancho y alto del widget label, en el fichero .Xdefaults se añadirán las siguientes entradas:

```
hola.etiqueta.width:  200
hola.etiqueta.height: 300
```

Tener en cuenta que cada vez que la aplicación se ejecuta, el fichero .Xdefaults se examina por lo que los cambios en dicho fichero no tienen efecto hasta que se ejecuta de nuevo la aplicación.

3.2.2 Configuración por Programa

Para cambiar un recurso cuando se crea un widget se utilizan los comandos tercero y cuarto de la rutina de creación como se vió en el programa `hola-motif.c`. Estos parámetros corresponden a un vector de pares $\langle nombre_recurso, valor_recurso \rangle$. El vector es de tipo

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Arg y tiene que ser de un tamaño suficiente para poder albergar todos los recursos que se asignen para un determinado widget. Para ir llenando los distintos elementos del vector se utiliza la rutina XtSetArg. Por ejemplo para cambiar el tamaño del widget label del programa ejemplo se añadirían antes de la creación del widget las siguientes líneas:

```
XtSetArg(args[n],XmNwidth,200); n++;
XtSetArg(args[n],XmNheight,300); n++;
```

Un comentario con respecto al uso de la rutina XtSetArg, es que la variable que hace de índice no se debe incrementar dentro del vector. Esto es porque XtSetArg es una macro que utiliza dos veces el vector y por tanto incrementaría dos veces el índice.

Después de creado el widget los recursos pueden ser cambiados de forma dinámica dentro del programa. Para ello se utiliza nuevamente un vector del tipo Arg y la rutina XtSetValues. Esta rutina tiene tres argumentos. El primero indica el widget al que se van a modificar los recursos. El segundo y tercero corresponden al vector y el número de elementos, al igual que en la macro XtSetArg. Por ejemplo si se quiere modificar el ancho y alto del widget label del programa hola-motif.c después de su creación, las instrucciones serían:

```
XtSetArg(args[n],XmNwidth,200); n++;
XtSetArg(args[n],XmNheight,300); n++;
XtSetValues(label,args,n);
```

Si un recurso tiene posibilidad de acceso para leer su valor (G), entonces se puede utilizar la rutina XtGetValues junto con un vector del tipo Arg para obtener el valor de los recursos. Cuando se utiliza esta rutina, en la macro XtSetArg se debe utilizar la dirección de una variable del mismo tipo que el recurso. Así para obtener el ancho y alto del widget del ejemplo será:

```
Dimension ancho,alto;

XtSetArg(args[n],XmNwidth,&ancho); n++;
XtSetArg(args[n],XmNheight,&alto); n++;
XtGetValues(label,args,n);
```

3.3 Cadenas de Caracteres y Fuentes

Motif define dos tipos de cadenas de caracteres: *String* y *XmString*. La primera se corresponde con un vector de caracteres, de igual forma que es definido en el lenguaje C. Este tipo de cadenas de caracteres se utilizan en los nombres de los widgets.

El tipo *XmString* es más complejo que el anterior tipo, ya que además de los caracteres a mostrar contiene información adicional. Esta información adicional está destinada a permitir la utilización de *OSF/Motif* en distintos lenguajes. Para ello, se incluye la fuente a utilizar para mostrar los caracteres y la dirección (en árabe se escribe de derecha a izquierda). Por tanto una XmString se corresponde a distintos segmentos, cada uno conteniendo información relativa a la fuente (*charset*), información sobre dirección y el texto en sí (figura 3.2).

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

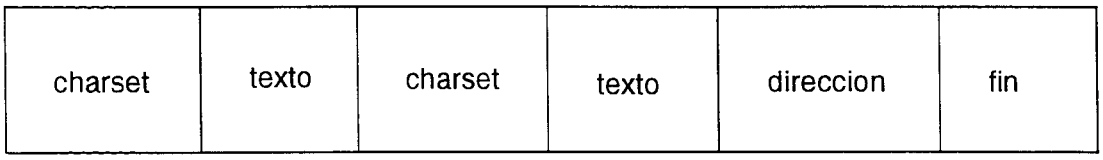

Figura 3.2: Estructura de una XmString

El charset es una forma de incluir fuentes en un programa. Con el charset, se asocia una o más fuentes a una etiqueta, bien por programa o el usuario. En concreto, existe una etiqueta que permite asociarle cualquier fuente que el usuario quiera. Esta etiqueta se denomina `XmSTRING_DEFAULT_CHARSET`, de forma que cualquier fuente elegida por el usuario coincidirá con este identificador. A continuación se verán algunas de las rutinas proporcionadas por Motif para la manipulación de las XmString. La rutina que crea una XmString a partir de una cadena de caracteres es `XmStringCreateLtoR()`. En el programa `hola-motif.c` ya se utilizó esta rutina para asignar el texto de la etiqueta que aparece en la ventana. El ejemplo se podría modificar ahora de la siguiente forma:

```
XmString xmcad;
String asciicad=''Hola Mundo''
xmcad=XmStringCreateLtoR(asciicad,XmSTRING_DEFAULT_STRING);
...
XtSetArg(args[n],XmNlabelString,xmcad); n++;
...
```

Para recuperar las distintas partes que componen una XmString existen diferentes rutinas. Así, para recuperar la cadena de caracteres que compone una XmString se utiliza la rutina `XmStringGetLtoR()`.

```
XmStringGetLtoR(xmcad,XmSTRING_DEFAULT_STRING,&asciicad);
```

Obtiene en `asciicad` la cadena de caracteres que se encuentra la XmString `xmcad`. Otras rutinas son:

```
Bool XmStringCompare(xmcad1, xmcad2)
XmString XmStringConcat(xmcad1, xmcad2)
XmString XmStringCopy(xmcad)
```

Como se desprende de sus nombres realizan la comparación de dos XmString, la concatenación de dos XmString y la copia de un XmString sobre otra; respectivamente. Un aspecto importante a tener en cuenta, es que cuando se crea una XmString, se utilizan una serie de recursos del sistema que se deben liberar. La liberación se realiza mediante la rutina `XmStringFree(xmcad)`, cuando una XmString ya no se va a utilizar más.

A continuación se verá como cambiar las fuentes para el texto que aparecen en las XmString. Para poder realizar esto, se define la fuente que se va a utilizar y luego crear una lista de fuentes (*Fontlist*) en un charset. De esta forma las XmString creadas con dicho charset, aparecerán con la fuente asociada al fontlist. Para seleccionar la fuente a

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

utilizar se utiliza la rutina `XLoadQueryFont` de la librería Xlib. Esta rutina utiliza como parámetros el display y el nombre de la fuente cargar. El display se obtiene con la macro `XtDisplay()` y el nombre de la fuente es cualquiera de las que existan en el sistema. Para ver las fuentes existentes se puede utilizar el comando

```
xlsfonts -fn ''*'' > fichero-temporal
```

que vuelca todas las fuentes disponibles en el sistema en el fichero `fichero-temporal`. A continuación se crea el fontlist asociado a un charset, utilizando la fuente que ha devuelto la función `XLoadQueryFont` y el charset `XmSTRING_DEFAULT_CHARSET`. Por último solo queda modificar el recurso `XmNfontList` del widget. Como ejemplo se muestra a continuación las instrucciones necesarias para cambiar el tipo de letra del mensaje en el programa `hola-motif.c`.

```
/* Añadir a las declaraciones */
  XFontStruct *font=NULL;
  XmFontList fontlist=NULL;
  char *nombre_fuente=NULL;
  ...
  nombre_fuente=''*6x10*'';
  font=XLoadQueryFont(XtDisplay(toplevel),nombre_fuente);
  fontlist=XmFontListCreate(font,XmSTRING_DEFAULT_CHARSET);
  ...
/* Añadir a los recursos de etiqueta */
  XtSetArg(args[n],XmNfontList,fontlist); n++;
```

Hacer notar que cuando se da el nombre de la fuente a cargar, se puede utilizar el carácter `*`. Este carácter funciona de la misma forma que al hacer un `ls`, es decir, se sustituye por cualquier cadena de caracteres.

Para cambiar la fuente del widget a partir del fichero Xdefaults se tienen que añadir la siguiente línea:

```
hola.etiqueta.fontList: "10x20"
```

En este caso es necesario que el nombre de la fuente a utilizar vaya entre comillas.

3.4 Callbacks

Para explicar el concepto de callback, se va a hacer uso del siguiente programa (`boton.c`), que crea un botón con una etiqueta.

```
#include <Xm/Xm.h>
#include <Xm/PushB.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,button;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

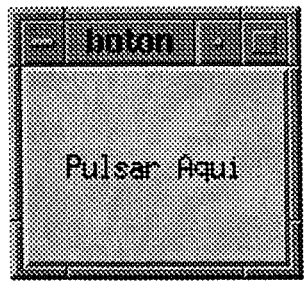

Figura 3.3: Progama "Boton"

```
void  main(int, char **);
void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,NULL,0);
  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Pulsar Aqui",
           char_set)); n++;
  button=XmCreatePushButton(toplevel,"boton",args,n);
  XtManageChild(button);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

Como se puede ver en el programa, la estructura es similar a la de `hola-motif.c`. Comienza inicializando la ventana principal de la aplicación (`toplevel`). A continuación, se establece el texto del botón y se crea. Luego se maneja y realiza el widget y por último se entra en el bucle para atender los eventos. Si se compila el programa y se ejecuta (figura 3.3), se puede comprobar que cuando se hace *click* con el ratón sobre el botón, cambia de color al igual que las sombras; dando la impresión que se hunde.

En el ejemplo anterior, el widget responde a una serie de eventos. Por ejemplo cambia de color cuando se hace *click*, o cambia de tamaño la ventana. Pero aparte de eso, el usuario no tiene más control sobre el widget. Si el usuario quisiera realizar alguna tarea cuando en un widget ocurre un evento, tiene que hacer uso de los denominados *Callbacks*. Un callback es una rutina que crea el usuario para que el programa realice determinadas acciones cuando ocurre un evento en un widget. En general, los callbacks definidos por el usuario se añaden a los que ya existen predeterminados para el evento en el widget. De forma que si el programador quiere que se realice alguna acción cuando se pulsa el botón, ésta se debe añadir a la de cambio de aspecto que ya se encuentra implementada. A continuación se muestra como se puede añadir una rutina que muestra un mensaje cada vez que se pulsa el botón. La rutina es la siguiente:

```
void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Boton pulsado\n");
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Los parámetros de los callbacks son tres. El primero de tipo `Widget` es el widget donde se ha producido el evento. El segundo es un dato que se pasa al callback. Este dato puede ser cualquiera que pueda ir en 4 bytes, un entero, un puntero a alguna estructura, etc. El último parámetro es un puntero a una "estructura callback". Esta estructura contiene un entero con la razón de porqué fue llamado el callback, ya que un mismo callback puede ser llamado para el mismo widget por distintos motivos; y un puntero a la estructura del evento que contiene toda la información relacionada con éste.

A continuación se debe añadir la rutina anterior a la lista de callbacks asociada al evento de pulsar del botón. Para ello se utiliza la siguiente rutina, después de haber manejado el widget.

```
XtAddCallback(button,XmNactivateCallback,maneja_boton,NULL);
```

La rutina anterior tiene cuatro parámetros. El primero es el widget al que se le asocia el callback. El segundo indica que evento activa el callback. El tercer parámetro es el nombre de la rutina, y el cuarto es el dato que se le va a pasar al callback como segundo parámetro. En el ejemplo anterior no se pasa ningún parámetro a la rutina. Los diferentes eventos a los que se les puede asociar un callback, aparecen entre los recursos del widget en la página de manual correspondiente. Así para el botón utilizado en el ejemplo estos eventos son: `activate`, `arm`, `disarm` y `destroy`. La descripción de cada uno de estos eventos así como la de la "estructura callback" aparece explicada en la página de manual.

A continuación se da un ejemplo de como pasar una cadena de caracteres (`cadena`) a la rutina.

```
XtAddCallback(button,XmNactivateCallback,maneja_boton,(XtPointer)cadena);
```

y dentro de la rutina se debe asigna la cadena como:

```
void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  char *mensaje=(char *)client_data;
  printf("Boton pulsado: %s\n",mensaje);
}
```

De la misma forma que se ha pasado una cadena de caracteres se puede pasar cualquier dato.

3.5 Jerarquía de los widgets

En la arquitectura de Motif al estar orientada a objeto, los widgets se agrupan en clases. Cada clase posee una serie de recursos que son heredados por las clases derivadas de ella. La superclase de la que derivan todos los widgets es la clase `Core`. De esta superclase se derivan las dos clases más importantes de widgets: `Composite` y `XmPrimitive` (figura 3.4). Los primeros son widgets que a su vez pueden contener a otros. Dentro de estos existen a su vez dos clases `Shell` y `XmManager`. Los widgets pertenecientes a la clase `XmPrimitive` son aquellos que solo pueden ser widgets hijos, es decir, no van a ser padres de ningún otro widget. A continuación se pasan a ver los widgets que existen dentro de las tres clases anteriores.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

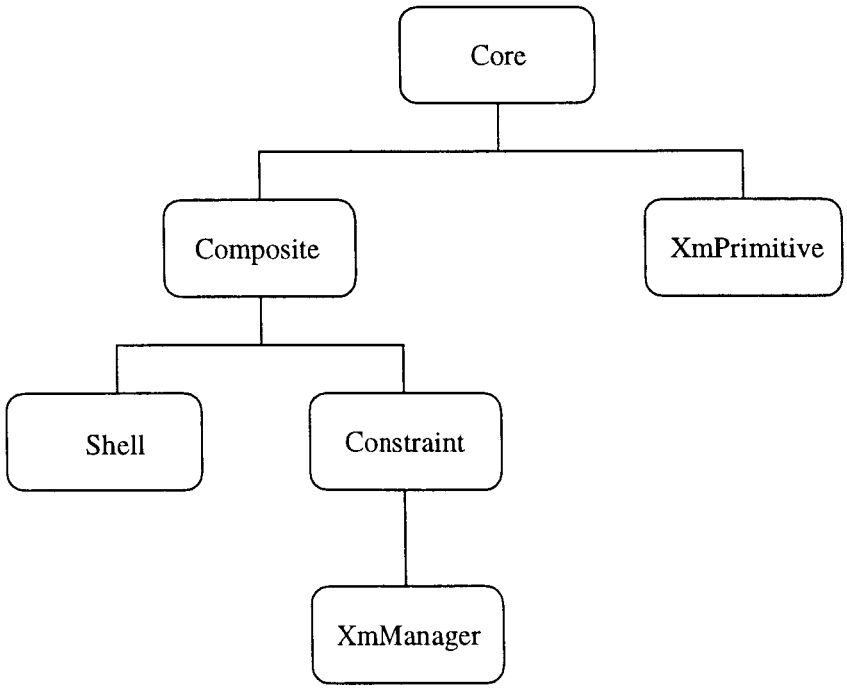

Figura 3.4: Nivel más alto del jerarquía de widgets

3.6 Widgets Shell

La principal función de estos widgets es la de servir de interfaz entre la aplicación y el sistema X-Window a través del manejador de ventanas. Los distintos tipos de widgets shell se pueden ver en la figura 3.5.

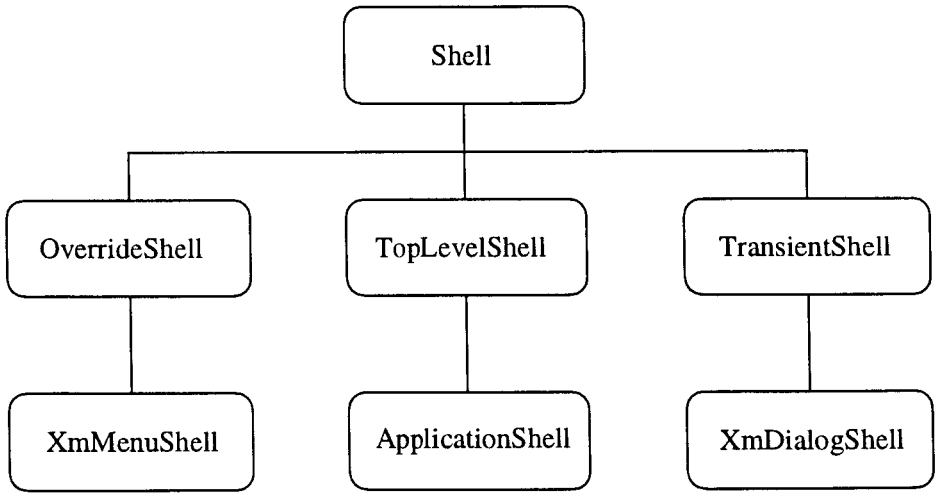

Figura 3.5: Jerarquía de widgets Shell

El shell `ApplicationShell` es el que se crea cuando se ejecuta la rutina `XtApp Initialize`. Este es el principal widget de una aplicación y no necesita un widget padre. Cuando este widget se destruye, toda la aplicación se destruye. Al ser el widget

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

principal, solo existe uno por aplicación. El shell *XmMenuShell* se utiliza en los menus. Como característica está la de no tener un marco para interactuar con el manejador de ventanas. Normalmente el diseñador no suele utilizar este tipo de shell directamente, aunque si fuera necesario existe la rutina `XmCreateMenuShell` que lo crea. Por último el tipo `XmDialogShell` es un shell Motif que se utiza como padre de todos los cuadros de diálogo. Normalmente se crea de forma automática cuando se crea un cuadro de diálogo, aunque se si desea crear directamente existe la rutina `XmCreateDialogShell`. Estos widgets vistos en este apartado normalmente no suelen ser utilizados directamente por los diseñadores ya que todos se crean de forma implícita al crear otros widgets, por ello no se entrará en más detalle.

3.7 Widgets Contenedores

3.7.1 Tablón

El widget `XmBulletinBoard` es el que menos acomoda a los widgets que va a contener, ya que no distribuye los mismos de forma automática ni los ajusta cuando cambia de tamaño. Por tanto es responsabilidad del programador, el asignar una posición a cada uno de los widgets que contiene y tener en cuenta cuando cambia de tamaño para recomponer la geometría de nuevo. Un programa ejemplo de un XmBulletinBoard con dos botones es el siguiente:

```
#include <Xm/Xm.h>
#include <Xm/BulletinB.h>
#include <Xm/PushB.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,tablon,boton1,boton2;

void main(int, char **);
void maneja_boton(Widget, XtPointer, XtPointer);

void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Boton pulsado\n");
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  char *cadena=NULL;

  toplevel=XtAppInitialize(&app_context,"tablon",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNwidth,200); n++;
  XtSetArg(args[n],XmNheight,50); n++;
  tablon=XmCreateBulletinBoard(toplevel,"tablon",args,n);
  XtManageChild(tablon);

  n=0;
  XtSetArg(args[n],XmNlabelString,
           XmStringCreateLtoR("Boton-1", char_set)); n++;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  XtSetArg(args[n],XmNx,10); n++;
  XtSetArg(args[n],XmNy,10); n++;
  boton1=XmCreatePushButton(tablon,"boton",args,n);
  XtManageChild(boton1);

  n=0;
  XtSetArg(args[n],XmNlabelString,
           XmStringCreateLtoR("Boton-2", char_set)); n++;
  XtSetArg(args[n],XmNx,100); n++;
  XtSetArg(args[n],XmNy,10); n++;
  boton2=XmCreatePushButton(tablon,"boton",args,n);
  XtManageChild(boton2);

  XtAddCallback(boton1,XmNactivateCallback,maneja_boton,NULL);
  XtAddCallback(boton2,XmNactivateCallback,maneja_boton,NULL);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

En el ejemplo anterior se pueden ver dos nuevos recursos que son comunes a todos los widgets, ya que son heredados de la clase `Core`, como son `XmNx` y `XmNy`, que indican la posición del widget. La salida es la que se puede ver en la figura 3.6. Si a esta ventana se le reduce el tamaño se puede ver como los botones pueden llegar a ocultarse.

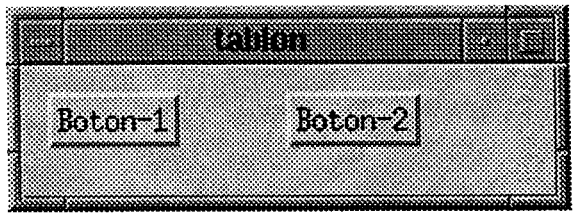

Figura 3.6: Programa basado en Bulletin Board

3.7.2 Formulario

Este widget (`XmForm`), a diferencia del Bulletin Board, si mantiene la relación espacial entre los widgets que contiene. De esta forma cuando se cambia de tamaño al widget XmForm, los widgets hijos también cambia de tamaño para mantener siempre el mismo aspecto. Para conseguir lo anterior, se utilizan unos recursos que se denominan `Attachment` (ligadura) en cada uno de los cuatro lados de los widgets hijos (arriba, abajo, derecho e izquierdo). Para indicar los lados existen los recursos `XmNtopAttachment`, `XmNbottomAttachment`, `XmNrightAttachment` y `XmNleftAttachment` respectivamente. Y a cada una de estas ligaduras se les puede dar uno de los siguientes valores:

- `XmATTACH_NONE` No se produce ninguna ligadura en el lado especificado del widget.
- `XmATTACH_FORM` Se une el lado especificado del widget con el lado correspondiente del XmForm.
- `XmATTACH_OPPOSITE_FORM` Se une el lado especificado del widget con el contrario del XmForm.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

- `XmATTACH_WIDGET` Se une el lado del widget con el lado opuesto del especificado del widget que se indica.
- `XmATTACH_OPPOSITE_WIDGET` Se une el lado especificado del widget con el mismo del widget que se indica.
- `XmATTACH_POSITION` El lado especificado del widget se pone en una posición que es relativa al lado correspondiente del XmForm y proporcional a la dimensión del lado especificado del widget.
- `XmATTACH_SELF` Se pone el widget en una posición que es proporcional al valor x o y del lado especificado, dividido por el ancho o el alto del XmForm.

Cuando una de las ligaduras especifica otro widget, es necesario indicar el widget al que se va a unir, para ello se asigna a uno de los siguientes recursos: `XmNtopWidget`, `XmNbottomWidget`, `XmNrightWidget` o `XmNleftWidget`, el widget con el que se va a unir. Estos recursos al igual que los vistos antes de las definiciones anteriores se heredan del XmForm, por lo que se asignan después de la creación de los widgets hijos del XmForm mediante la rutina `XtSetValues`. La mejor forma de ver la función de cada uno de las anteriores ligaduras es con un ejemplo y luego hacer una serie de cambios en el mismo para observar como cambia la distribución de los widgets. En el siguiente ejemplo se muestra un programa con un XmForm y cuatro botones como hijos. En la figura 3.7 se muestra la apariencia del programa cuando se ejecuta y en la 3.8 después de aumentar el tamaño de la ventana.

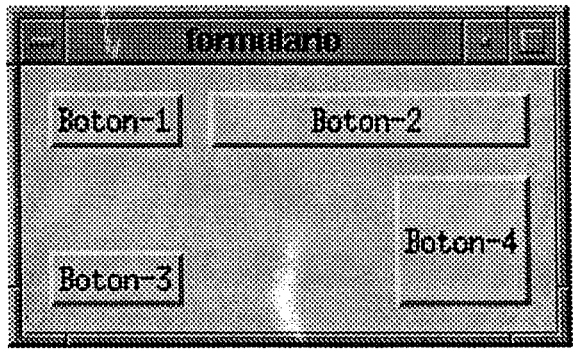

Figura 3.7: XmForm con cuatro botones

A continuación se muestra la parte de programa `formulario.c` donde se crea el widget XmForm y se asignan valores a las ligaduras de los cuatro botones que contiene el widget XmForm.

```
n=0;
XtSetArg(args[n],XmNwidth,200); n++;
XtSetArg(args[n],XmNheight,100); n++;
XtSetArg(args[n],XmNverticalSpacing,10); n++;
XtSetArg(args[n],XmNhorizontalSpacing,10); n++;
formulario=XmCreateForm(toplevel,"formulario",args,n);
XtManageChild(formulario);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

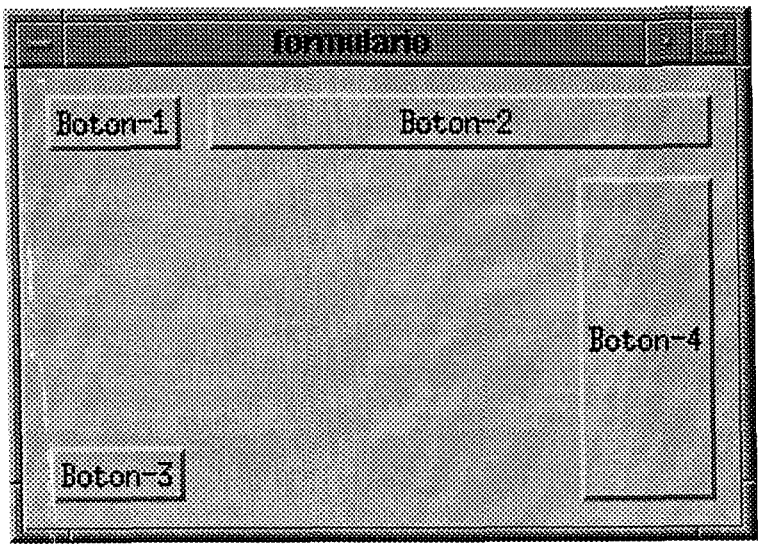

Figura 3.8: XmForm con cuatro botones despues de cambiar tamaño

```
...
n=0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
XtSetValues(boton1,args,n);

n=0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
XtSetArg(args[n],XmNleftWidget,boton1); n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET); n++;
XtSetValues(boton2,args,n);

n=0;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
XtSetValues(boton3,args,n);

n=0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
XtSetArg(args[n],XmNtopWidget,boton2); n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
XtSetValues(boton4,args,n);
```

Hacer notar que si no especifica nada, los widgets contenidos en un form no dejan espacio entre ellos, si están unidos. Para especificar el espacio que queda entre ellos se utilizan los recursos `XmNverticalSpacing` y `XmNhorizontalSpacing` para el espaciado vertical y horizontal respectivamente.

3.7.3 Apilados

Este tipo de widget (`XmPanedWindow`) distribuye los widgets hijos en ventanas apiladas verticalmente. Cada ventana se puede identificar por medio de un separador y un deslizante, que sirve para ajustar el tamaño de cada ventana. En la figura 3.9 se muestra

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

una ventana con un paned widget que contiene 2 botones. Por defecto se muestran los separadores, para que no aparezcan se debe poner el recurso `XmNseparatorOn` a `False`. El deslizante puede colocarse en cualquier parte del separador. Para indicar la posición se utiliza el recurso `XmNsashIndent`, si se le asigna un valor positivo indica un desplazamiento a partir del lado izquierdo. Si el valor es negativo el desplazamiento desde el lado derecho. Por defecto el valor es -10. El ancho y alto de este deslizante es dado mediante `XmNsashWidth` y `XmNsashHeight`.

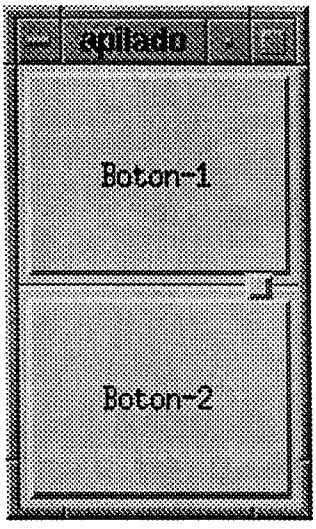

Figura 3.9: Widget Paned con dos botones

Un fragmento del programa `apilado.c` donde se define el widget y dos botones hijos es el siguiente

```
n=0;
apilado=XmCreatePanedWindow(toplevel,"apilado",args,n);
XtManageChild(apilado);

n=0;
XtSetArg(args[n],XmNwidth,100); n++;
XtSetArg(args[n],XmNheight,75); n++;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-1", char_set)); n++;
boton1=XmCreatePushButton(apilado,"boton",args,n);
XtManageChild(boton1);

n=0;
XtSetArg(args[n],XmNwidth,100); n++;
XtSetArg(args[n],XmNheight,75); n++;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-2", char_set)); n++;
boton2=XmCreatePushButton(apilado,"boton",args,n);
XtManageChild(boton2);
```

3.7.4 Fila Columna

Este tipo de widget (`XmRowColumn`) es bastante utilizado, sobre todo en menú. La distribución de los hijos de este widget se realiza en función del valor dado a los recursos. Así los widgets hijos se pueden agrupar en filas o columnas y si se quiere en cajas del mismo

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

tamaño. El ejemplo de este tipo de widget se puede ver en la figura 3.10, correspondiente al programa `fila-columna.c`. En el ejemplo se han colocado seis botones sobre un widget XmRowColumn.

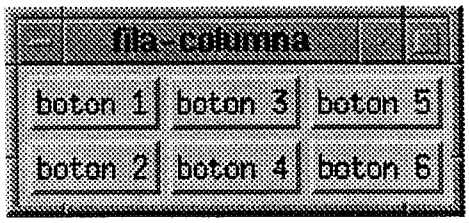

Figura 3.10: Widget RowColumn

La distribución de los widgets viene dada por el recurso `XmNorientation` que en el caso del ejemplo está al valor por defecto que es `XmVERTICAL`. Con este valor los widgets hijos se agrupan por columnas en función del valor dado al recurso `XmNnumColumns`. A continuación se muestra la parte del programa donde se crea el widget XmRowColumn.

```
n=0;
XtSetArg(args[n],XmNnumColumns,3); n++;
XtSetArg(args[n],XmNpacking,XmPACK_COLUMN); n++;
filacol=XmCreateRowColumn(toplevel,"filacol",args,n);
XtManageChild(filacol);
```

En el ejemplo anterior el recurso `XmNpacking` sirve para que todos los widgets hijo tenga el mismo tamaño. Otros dos valores que puede tener este recurso son: `XmPACK_NONE` y `XmPACK_TIGHT`. Si en el ejemplo se asigna al recurso `XmNorientation` el valor `XmHORIZONTAL`, el resultado que se obtiene es el que se muestra en la figura 3.11.

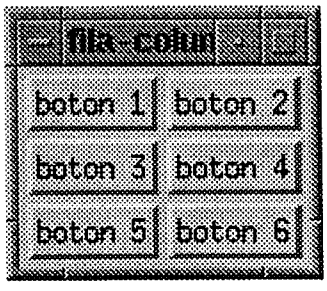

Figura 3.11: Widget RowColumn con orientación horizontal

Un detalle que se aprecia en el resultado es que si el numero de columnas se hizo igual a 3, por que aparecen solo 2. La solución viene del hecho que el recurso `XmNnumColumns` hace referencia al numero de columnas si la orientación es vertical y al numero de filas si la orientación el horizontal.

3.7.5 Ventana Deslizante

La ventana deslizante (`XmScrolledWindow`) consiste en una zona visible, una barra de desplazamiento horizontal y otra vertical. La barras deslizantes son visibles dependiendo

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

de la información que se muestra en la ventana deslizante y del valor de ciertos recursos. La figura 3.12 muestra la salida del programa `vent-deslizante.c`, que crea una ventana deslizante con una serie de etiquetas en su interior.

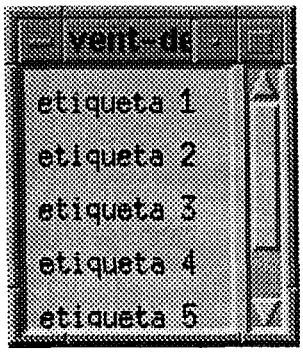

Figura 3.12: Widget Scrolled Window conteniendo 6 etiquetas

El programa `vent-deslizante.c` consta de un frame que va a ser el que contiene el Scrolled Window. Las etiquetas van a estar en un widget RowColumn. A continuación se muestra la parte del programa que crea la ventana deslizante.

```
n=0;
frame=XmCreateFrame(toplevel,"frame",args,n);
XtManageChild(frame);

n=0;
XtSetArg(args[n],XmNwidth,100); n++;
XtSetArg(args[n],XmNheight,100); n++;
XtSetArg(args[n],XmNscrollingPolicy,XmAUTOMATIC); n++;
ventanad=XmCreateScrolledWindow(frame,"vent_desl",args,n);
XtManageChild(ventanad);
```

En este caso, se ha creado un widget *frame* (se verá en la sección 3.7.7) para dar la impresión de tridimensionalidad, y como hijo de éste se ha creado la ventana deslizante. Un recurso de la ventana deslizante es la `XmNscrollingPolicy` que es asignado al valor `XmAUTOMATIC`. Con este valor es la ventana deslizante la que se encarga de gestionar las barras deslizantes, creándolas y activándolas cuando el widget hijo no puede ir completo en el área de trabajo. El valor por defecto es `XmAPPLICATION_DEFINED`, en cuyo caso es la aplicación la encargada de gestionar todo lo relacionado con las barras deslizantes. Si se utiliza la opción anterior, se indican los distintos elementos de la ventana deslizante mediante la siguiente rutina.

```
XmScrolledWindowSetAreas(ventanad,NULL,NULL,NULL);
```

que tiene como parámetros: la ventana deslizante, la barra horizontal, la barra vertical y el área visible. En el ejemplo al ser la ventana deslizante la encargada de las barras, no se especifica nada, asignándoles el valor `NULL`.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

3.7.6 Ventana Principal

Motif proporciona este widget (`XmMainWindow`) como plantilla para la ventana principal de cualquier aplicación. El diseño de esta ventana sigue la propuesta de la *Guía de Estilo OSF/Motif*, la cual establece una barra de menú, dispuesta horizontalmente, en la parte superior de la ventana. A continuación vendría una ventana de comandos y luego el área de trabajo, que se corresponde con una ventana deslizante. Por último se estaría la ventana de mensajes, si la hubiera. La ventana de comandos se puede poner debajo del área de trabajo asignando el valor `XmCOMMAND_BELOW_WORKSPACE` al recurso `XmNcommandWindowLocation`. Entre cada dos ventanas se puede hacer que exista un separador. Esto lo realiza el widget XmMainWindow de forma automática, asignando el valor verdadero al recurso `XmNshowSeparator`. Aparte de los recursos que posee la ventana deslizante para las barras de deslizantes, el XmMainWindow posee los siguientes recursos que contienen el identificador de los distintos widgets que posee: `XmNcommandWindow`, `XmNmenuBar` y `XmNmessageWindow`. En la figura 3.13 se muestra la salida del programa `ventana-pral.c` que crea una aplicación con un widget XmMainWindow.

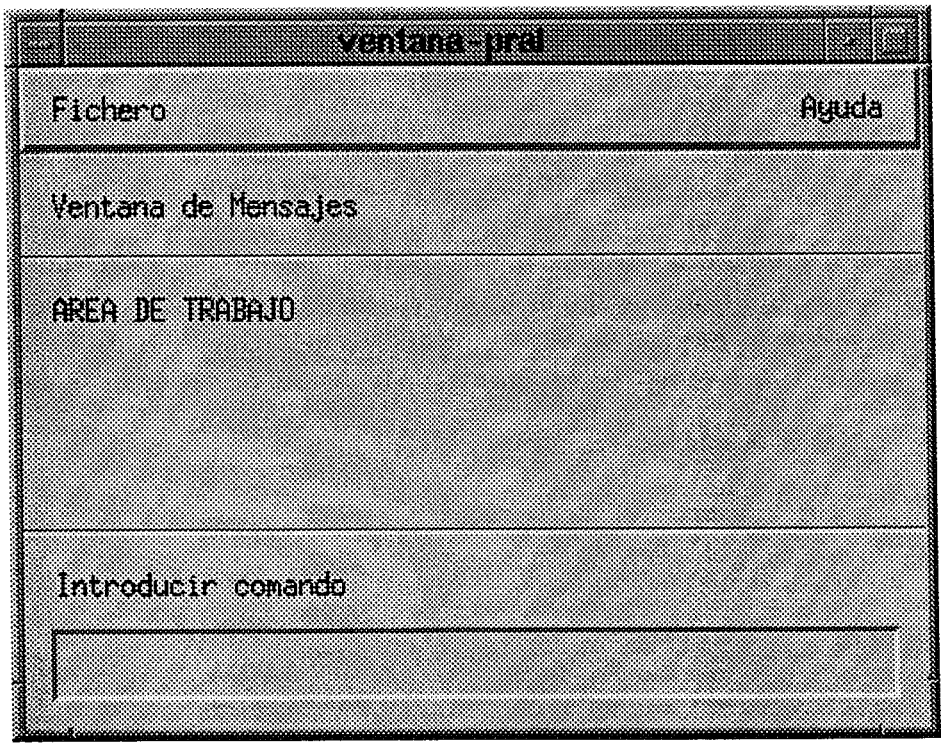

Figura 3.13: Ventana principal de las aplicaciones en Motif

La ventana que se muestra en la figura 3.13 es el resultado de ejecutar el programa `ventana-pral.c`. La parte del código correspondiente a la creación y manejo del widget XmMainWindow es:

```
n=0;
XtSetArg(args[n],XmNshowSeparator,True); n++;
vent_pral=XmCreateMainWindow(toplevel,"vent_pral",args,n);
XtManageChild(vent_pral);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
/* Creacion de la barra de menu principal */
  menu=crea_menu_pral(vent_pral);
/* Creacion de la ventana de comandos */
  comando=crea_comandos(vent_pral);
/* Area de trabajo */
  n=0;
  XtSetArg(args[n],XmNwidth,100); n++;
  XtSetArg(args[n],XmNheight,100); n++;
  area_trabajo=XmCreateBulletinBoard(vent_pral,"tablon",args,n);
  XtManageChild(area_trabajo);
```

La creación de la barra de menú se comentará más adelante en la sección 3.9.1. La ventana de comandos corresponde a un widget XmBulletinBoard con un campo de texto, donde se introduce los comandos del usuario. El área de trabajo se ha creado mediante un XmBulletinBoard, aunque se pueden utilizar otro widget contenedor. Para la asignar la funcionalidad a cada uno de los componentes de la ventana principal se utiliza la siguiente función:

```
XmMainWindowSetAreas(vent_pral,menu,comando,NULL,NULL,area_trabajo);
```

Los dos elementos que se encuentran a `NULL` corresponden a los widgets de las barras deslizantes que no se van a utilizar. Un aspecto a tener en cuenta es que en la anterior rutina no existe un parámetro para la ventana de mensajes. Esto es debido a que esta ventana se introdujo después de la versión 1.1 de Motif, a la que pertenece la anterior rutina. Por tanto para asignar la ventana de mensajes se debe hacer mediante el uso del recurso `XmNmessageWindow`. A este recurso se le asigna el identificador del widget que se ha utilizado como ventana de mensaje, como se puede ver en el siguiente código:

```
mensaje=crea_vent_mensaje(vent_pral);
n=0;
XtSetArg(args[n],XmNmessageWindow,mensaje); n++;
XtSetValues(vent_pral,args,n);
```

3.7.7 Marco

El marco (`XmFrame`), es un widget que se utiliza para encerrar a los widgets hijos en un marco con una sombra alrededor, dando sensación de tridimensionalidad. Los recursos mas utilizados son `XmNshadowType` que da el tipo de sombra, `XmNmarginHeight` y `XmNmarginWidth` para asignar el alto y ancho de margen entre el widget hijo y la sombra y `XmNshadowThickness` que es el ancho en pixels de la sombra.

Los widget frames pueden tener un titulo, utilizando un widget hijo (normalmente una etiqueta) para ese cometido. El recurso utilizado para especificar un widget hijo como titulo es `XmNframeChildType`, mediante la asignación del valor `XmFRAME_TITLE_CHILD`. En la figura 3.14 se muestra la salida del programa `marco.c`. Este programa utiliza un frame para incluir el form, visto en la sección de formularios, y una etiqueta. La salida se puede comparar con la obtenida en la figura 3.7.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

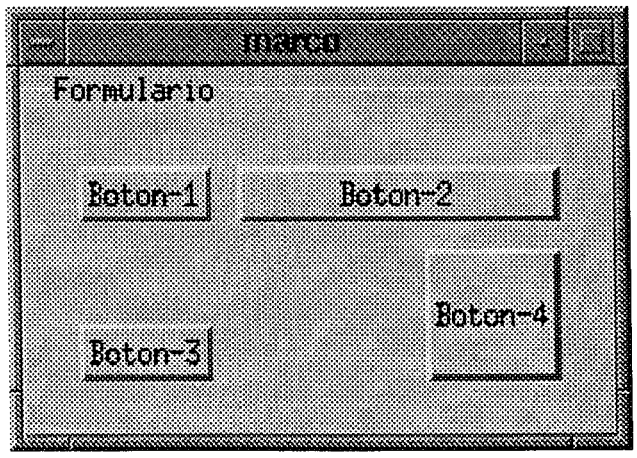

Figura 3.14: Widget XmForm con un marco

A continuación se muestra la parte del programa donde se crea el frame y la etiqueta como titulo del marco.

```
n=0;
XtSetArg(args[n],XmNmarginWidth,10); n++;
XtSetArg(args[n],XmNmarginHeight,10); n++;
marco=XmCreateFrame(toplevel,"frame",args,n);
XtManageChild(marco);

n=0;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Formulario",
        char_set)); n++;
XtSetArg(args[n],XmNframeChildType,XmFRAME_TITLE_CHILD); n++;
etiqueta=XmCreateLabel(marco,"etiqueta",args,n);
XtManageChild(etiqueta);

n=0;
XtSetArg(args[n],XmNwidth,200); n++;
XtSetArg(args[n],XmNheight,100); n++;
XtSetArg(args[n],XmNverticalSpacing,10); n++;
XtSetArg(args[n],XmNhorizontalSpacing,10); n++;
XtSetArg(args[n],XmNframeChildType,XmFRAME_WORKAREA_CHILD); n++;
formulario=XmCreateForm(marco,"formulario",args,n);
XtManageChild(formulario);
```

Debido a que el formulario va a ser el widget principal del frame, se asigna al recurso `XmNframeChildType` el valor `XmFRAME_WORKAREA_CHILD`. Hay que indicar que este recurso que indica el tipo de hijo, a pesar de ser un recurso del frame, se deben especificar para los widgets hijos, ya que son estos los que lo heredan.

3.7.8 Contenedor

Este tipo de widget solo puede tener como hijos a widgets de la clase `XmIconGadget` o de alguna subclase. Los IconGadgets se utilizan en las aplicaciones para mostrar gráficos o texto. El widget de tipo contenedor `XmContainer` se puede utilizar para visualizar los IconGadgets en diferentes formatos, seleccionarlos y manipularlos.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

En el contenedor los IconGadgets se pueden disponer de diferentes modos, según el valor que se le asigne al recurso `XmNlayoutState`. Uno de estos modos es el `XmOUTLINE`, que permite mostrar la información estableciendo relaciones jerárquicas de forma visual. Como ejemplo puede ser la información sobre directorios, subdirectorios y ficheros de un determinado volumen. Otro modo similar al anterior es el `XmDETAIL`, que además permite mostrar información adicional, en el caso anterior, puede ser tamaño de los ficheros, fechas de creación, etc. El otro modo es el `XmSPATIAL`, donde la información se puede mostrar en forma de rejilla, o en posiciones específicas. Una aplicación de este último modo puede ser para representar estaciones de trabajo mediante IconGadgets, sobre un fondo que se corresponda a la topología de una red departamental. En este modo, los IconGadgets pueden ser arrastrados y puestos en otras posiciones del widget XmContainer.

En la figura 3.15 se muestra un widget del tipo XmContainer con distribución `XmDETAIL`. Para cada puesto se da la dirección y la dirección de correo electrónico. El primero se encuentra desplegado, mientras que la segunda entrada si se pulsa la flecha se desplegara la información que depende de éste (figura 3.16). El despliege de un hijo se puede establecer mediante el recurso `XmNoutlineState`. En el caso de hijo "Del. Canarias" el recurso anterior se ha puesto a `XmEXPANDED`, con lo cual aparece desplegado. En el caso del widget "Del. Extremadura" no se ha asignado ningún valor con lo que mantiene el que tiene por defecto que es `XmCOLLAPSED`, no mostrando los hijos.

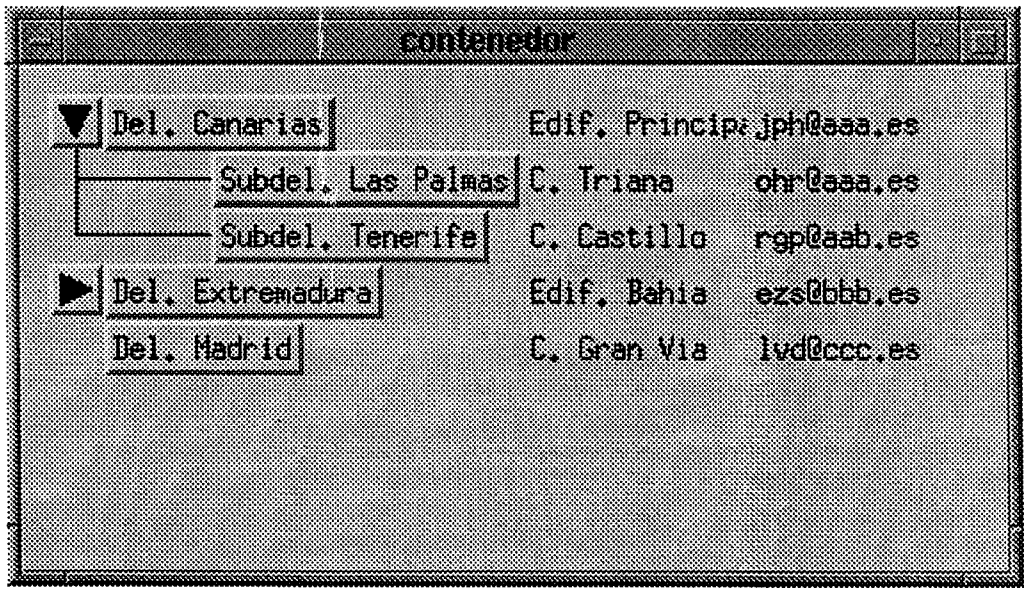

Figura 3.15: Widget Container

El programa que crea las ventanas anteriores es `contenedor.c` y el fragmento donde se crea el contenedor es el siguiente:

```
n=0;
XtSetArg(args[n],XmNlayoutType,XmDETAIL); n++;
XtSetArg(args[n],XmNselectionPolicy,XmBROWSE_SELECT); n++;
XtSetArg(args[n],XmNautomaticSelection,XmNO_AUTO_SELECT); n++;
XtSetArg(args[n],XmNentryViewType,XmLARGE_ICON); n++;
contenedor=XmCreateContainer(frame,"contenedor",args,n);
XtManageChild(contenedor);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

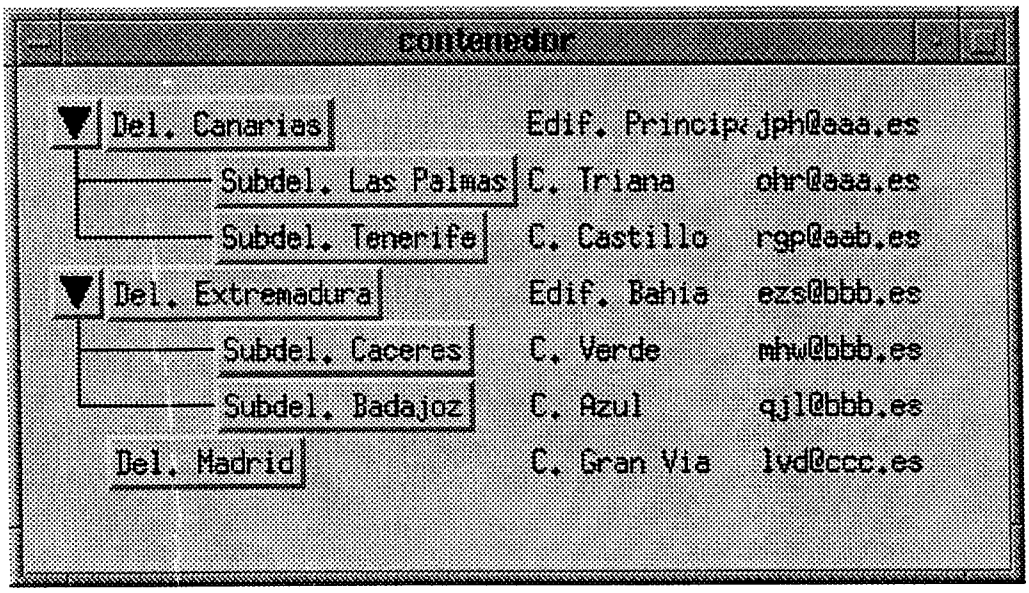

Figura 3.16: Widget Container con todos las entradas desplegadas

Para crear las entradas, es necesario indicar si va a ser una entrada de nivel cero, es decir, no asociada a otra entrada; o bien si tiene una dependencia jerárquica de otra entrada. En el primer caso no es necesario indicar ningún tipo de recurso, pero en el segundo caso, es necesario indicar cual es la entrada que le precede en la jerarquía. Para indicar esta precedencia se utiliza el recurso `XmNentryParent`, como se aprecia en la siguientes instrucciones del programa.

```
xmstr[0]=XmStringCreateLtoR("C. Triana", char_set);
xmstr[1]=XmStringCreateLtoR("ohr@aaa.es", char_set);

n=0;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Subdel. Las Palmas",
        char_set)); n++;
XtSetArg(args[n],XmNentryParent,entrada1); n++;
XtSetArg(args[n],XmNdetail,xmstr); n++;
XtSetArg(args[n],XmNdetailCount,2); n++;
entrada11=XmCreateIconGadget(contenedor,"entrada",args,n);
XtManageChild(entrada11);
```

Al ser la distribución del tipo `XmDETAIL`, es necesario indicar para cada entrada que datos adicionales deben aparecer. Esto se realiza mediante los recursos: `XmNdetail` y `XmNdetailCount`. El primero es un vector de XmString que contiene la información y el segundo indica cuantos elementos posee el anterior vector.

3.7.9 Block de Notas

El block de notas (`XmNotebook`) es un widget que tiene la apariencia de una libreta. Por tanto sus widgets hijos se tratan como las hojas, los marcadores o los nombres de hojas. Mediante el uso del recurso `XmNnotebookChildType`, que heredan los hijos, se puede dar a los distintos hijos el tratamiento antes mencionado. Así cuando el recurso anterior

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

para un determinado hijo posee el valor `XmPAGE`, este widget es tratado como una página. Como en cualquier block de notas, independiente del número de páginas que posea, solo se muestra una cada vez. La página que se muestra es la que coincide con el valor de recurso `XmNcurrentPageNumber`. Para que dicha coincidencia se pueda producir, cada widget del tipo hoja posee un recurso que indica que número de hoja tiene en la libreta. Este numero de hoja viene dado por el recurso `XmNpageNumber`. De esta forma para visualizar el contenido de un notebook en una aplicación se puede ir cambiando el valor de `XmNcurrentPageNumber`.

Otra posibilidad para ver el contenido de un notebook es asignar a cierta páginas el estado de tabuladores, mediante el recurso `XmNnotebookChildType` con los valores `XmMAJOR_TAB` o `XmMINOR_TAB`. De esta forma mediante esta asignación se puede acceder a las páginas correspondientes de forma directa, de la misma manera que se hace en un libro mediante las "solapas". El programa `block.c` implementa una pequeña aplicación mediante este tipo de widget. En esta aplicación se muestran marcas de coches, a las que se pueden acceder de forma rápida por procedencia o estilo de coche. En la figura 3.17 se muestra la salida de este programa.

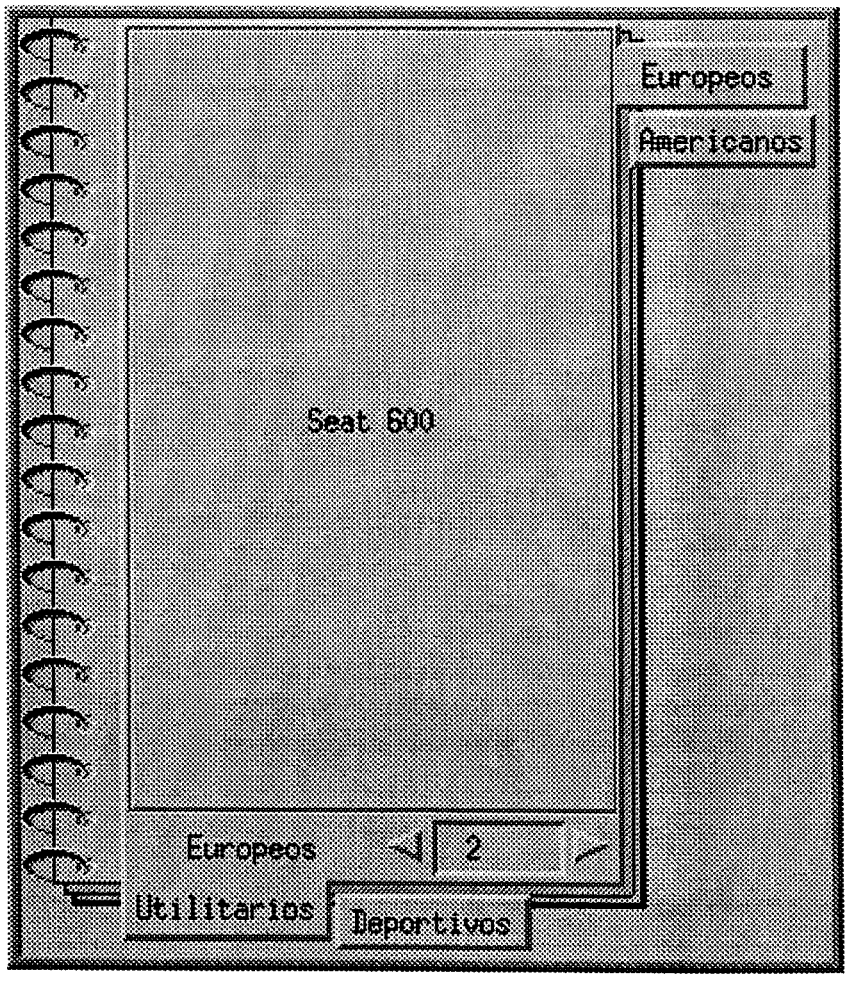

Figura 3.17: Aplicación basada en el widget Notebook

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Otra función que puede tener un widget hijo, es la de descripción de una página, para ello al recurso XmNnotebookChildType se le asigna el valor XmSTATUS_AREA. En la figura 3.17, aparece en la parte inferior de la página el nombre de dicha página. Cada vez que se cambia de página se produce el callback denominado XmNpageChangedCallback, de forma que el usuario puede asignar algún procedimiento al evento de cambio de página. A continuación se muestra algunos fragmentos del código del programa block.c donde se muestra la creación del notebook y de los distintos tipos de hijos que puede tener.

```
...
char *cadena[PAG_BLOCK+1]={"vacio",
                           "COCHES",
                           "Seat 600",
                           "VW escarabajo",
                           "Renault",
                           "Porche",
                           "BMW",
                           "Ferrari",
                           "Ford",
                           "Cadillac",
                           "Jeep",
                           "Pontiac"
                          };
...
n=0;
XtSetArg(args[n],XmNwidth,300); n++;
XtSetArg(args[n],XmNheight,350); n++;
block=XmCreateNotebook(toplevel,"block",args,n);
XtManageChild(block);
for (i=1;i<=PAG_BLOCK;i++) {
  n=0;
  XtSetArg(args[n],XmNnotebookChildType,XmPAGE); n++;
  XtSetArg(args[n],XmNpageNumber,i);
  frame=XmCreateFrame(block,"frame",args,n);
  XtManageChild(frame);
  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR(cadena[i],
           char_set)); n++;
  label=XmCreateLabel(frame,"label",args,n);
  XtManageChild(label);
}
n=0;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Europeos",
         char_set)); n++;
XtSetArg(args[n],XmNnotebookChildType,XmSTATUS_AREA); n++;
XtSetArg(args[n],XmNpageNumber,2); n++;
label=XmCreateLabel(block,"label",args,n);
XtManageChild(label);
n=0;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Europeos",
         char_set)); n++;
XtSetArg(args[n],XmNnotebookChildType,XmMAJOR_TAB); n++;
XtSetArg(args[n],XmNpageNumber,1); n++;
boton=XmCreatePushButton(block,"boton",args,n);
XtManageChild(boton);
n=0;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Utilitarios",
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
        char_set)); n++;
XtSetArg(args[n],XmNnotebookChildType,XmMINOR_TAB); n++;
XtSetArg(args[n],XmNpageNumber,2); n++;
boton=XmCreatePushButton(block,"boton",args,n);
XtManageChild(boton);
```

3.8 Controles

Los widgets que se van a ver en esta sección corresponden a los de la clase `XmPrimitive` (figura 3.18). Estos widgets son los que conforman la interfaz con el usuario propiamente dicha, porque son los widgets con los que el usuario interactúa.

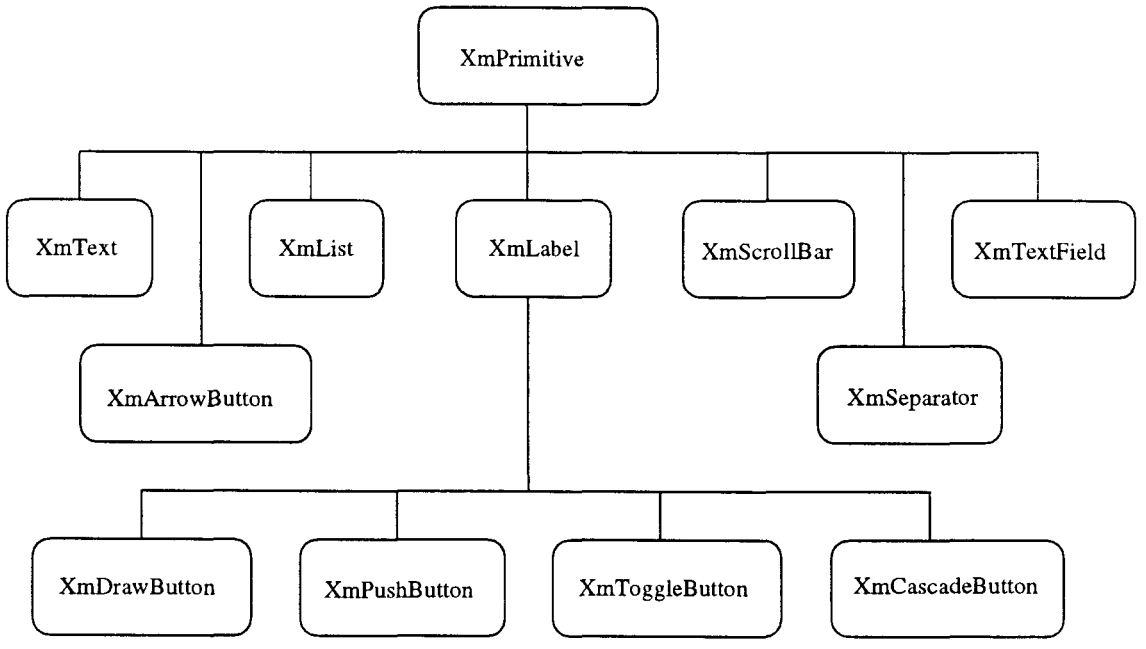

Figura 3.18: Widgets de tipo XmPrimitive

3.8.1 Etiquetas y Separadores

Las etiquetas (`XmLabel`) ya han sido utilizadas en varios ejemplos. Como se ha visto hasta ahora, su función es para mostrar textos estáticos. La fuente con que se muestra el texto es la que se define mediante el recurso `XmNfontList`, y que previamente se ha creado dentro de un charset. Aparte de los recursos ya vistos en ejemplos anteriores, existen dos que permiten modificar la posición del texto en el widget. Estos recursos son `XmNaligment` y `XmNstringDirection`. Otro recurso que puede ser interesante es `XmNrecomputeSize`, que hace que el widget intente mantenerse lo más grande posible para contener el texto. En la figura 3.19 se pueden ver tres widgets XmLabel con el mismo ancho pero con distinta justificación cada uno.

Las líneas de código que se muestran, pertenecen al programa `etiqueta.c` cuya salida es la que se muestra en la figura 3.19. Éste crea tres widgets XmLabel con distinta

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

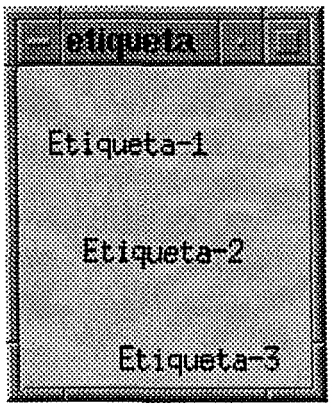

Figura 3.19: Widgets Labels con distinta justificación

justificación como hijos de un widget XmBulletinBoard. Las líneas corresponden a la creación y manejo de la segunda etiqueta.

```
n=0;
XtSetArg(args[n],XmNlabelString,
         XmStringCreateLtoR("Etiqueta-2", char_set)); n++;
XtSetArg(args[n],XmNx,1); n++;
XtSetArg(args[n],XmNy,60); n++;
XtSetArg(args[n],XmNwidth,90); n++;
XtSetArg(args[n],XmNalignment,XmALIGNMENT_CENTER); n++;
etiqueta2=XmCreateLabel(tablon,"etiqueta",args,n);
XtManageChild(etiqueta2);
```

Los separadores (`XmSeparator`) se utilizan para crear una separación entre grupos de controles. Los separadores pueden ser horizontales o verticales. La orientación viene dada por el recurso `XmNorientation`, que puede tomar los valores `XmVERTICAL` o `XmHORIZONTAL`. El tipo de línea utilizada en el separador se define en el recurso `XmNsepa ratorType`, y puede ser simple (defecto), doble, simple punteada, etc. Un ejemplo de la utilización de este tipo de widget se puede ver en la figura 3.20. La parte de código que crea el separador en el programa `botones.c` es:

```
n=0;
separador=XmCreateSeparator(formulario,"separador",args,n);
XtManageChild(separador);

n=0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
XtSetArg(args[n],XmNtopWidget,boton1); n++;
XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
XtSetValues(separador,args,n);
```

3.8.2 Botones

Dentro de este epígrafe, se van a ver los distintos tipos de botones que están definidos en *OSF/Motif*. Los cinco tipos son: *PushButtons*, *DrawnButtons*, *ArrowButtons*, *CascadeButtons* y *ToggleButtons*.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

PushButton

Este tipo de botones ya ha sido utilizado en diversos ejemplos en este texto. Al mismo se le puede asignar callbacks cuando se activan o cuando se arman o desarman, por medio de los recursos `XmNactivateCallback`, `XmNarmCallback` y `XmNdisarmCallback`. Estos botones tienen como característica que cambian su apariencia cuando son pulsados. En la figura 3.20 se pueden ver los distintos tipos de botones. Las siguientes líneas de código corresponden a la creación del PushButton en el programa `botones.c`.

```
n=0;
XtSetArg(args[n],XmNheight,58); n++;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Push Button",
         char_set)); n++;
boton1=XmCreatePushButton(formulario,"boton",args,n);
XtManageChild(boton1);
XtAddCallback(boton1,XmNactivateCallback,maneja_boton1,NULL);
n=0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
XtSetValues(boton1,args,n);
```

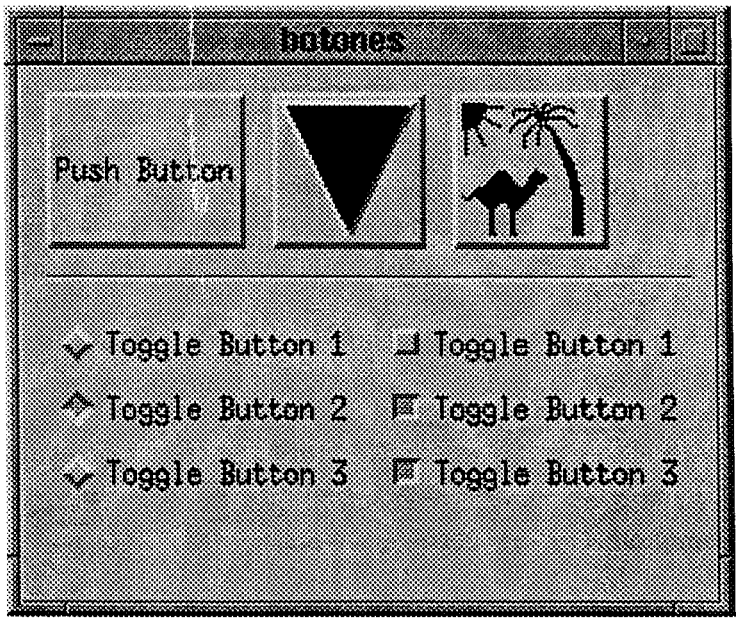

Figura 3.20: Ventana con diferentes tipos de botones

DrawnButton

Este tipo de botones es muy similar al anterior, con la diferencia de mostrar un dibujo en lugar de un texto. Este botón por defecto muestra un área vacía rodeada de una sombra que cambia de apariencia cuando se pulsa y es responsabilidad del usuario asignarle un dibujo. La apariencia del botón se puede cambiar para que aparezca como hundido

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

cuando se ha activado con el ratón, y hacia arriba cuando no se ha pulsado. Para obtener el resultado anterior se utiliza el recurso XmNpushButtonEnabled. Cuando el botón se cambia de tamaño o se pone otra ventana encima, se notifica a la aplicación por medio de los callbacks XmNresizeCallback y XmNexposeCallback respectivamente. De forma que es tarea de la aplicación redibujar cada vez que uno de los anteriores eventos ocurra. Hay que tener cuidado de no dibujar en lo que son las sombras del widget. En la figura 3.20 se puede ver un drawnbutton. Las siguientes líneas son las que corresponden en el programa botones.c a la creación del drawnbutton.

```
display=XtDisplay(toplevel);
XReadBitmapFile(display,DefaultRootWindow(display),fichero,&ancho,&alto,
                &camello,&x_hot,&y_hot);
n=0;
XtSetArg(args[n],XmNwidth,58); n++;
XtSetArg(args[n],XmNheight,58); n++;
XtSetArg(args[n],XmNlabelType,XmPIXMAP); n++;
XtSetArg(args[n],XmNpushButtonEnabled,True); n++;
boton3=XmCreateDrawnButton(formulario,"boton",args,n);
XtManageChild(boton3);
XtAddCallback(boton3,XmNactivateCallback,maneja_boton3,NULL);
n=0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
XtSetArg(args[n],XmNleftWidget,boton2); n++;
XtSetArg(args[n],XmNlabelPixmap,camello); n++;
XtSetValues(boton3,args,n);
```

El código es muy similar al de creación del resto de los botones, con la excepción de que en este caso es necesario generar el dibujo que va a aparecer en el botón. El dibujo es un pixmap que se genera a partir de un fichero que contiene un bitmap mediante la rutina XReadBitmapFile.

ArrowButton

Este tipo de botones no tiene etiqueta, ya que solo muestra un gráfico con una flecha sombreada (figura 3.20). La dirección de la flecha viene dada por el recurso XmNarrowDirection. El comportamiento es similar al de los botones vistos anteriormente con callbacks para la activación, armado y desarmado. A continuación se muestran las líneas con que se crea un widget de este tipo en el programa botones.c.

```
n=0;
XtSetArg(args[n],XmNwidth,58); n++;
XtSetArg(args[n],XmNheight,58); n++;
XtSetArg(args[n],XmNarrowDirection,XmARROW_DOWN); n++;
boton2=XmCreateArrowButton(formulario,"boton",args,n);
XtManageChild(boton2);
XtAddCallback(boton2,XmNactivateCallback,maneja_boton2,NULL);
n=0;
XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
XtSetArg(args[n],XmNleftWidget,boton1); n++;
XtSetValues(boton2,args,n);
```

ToggleButton

Estos botones se utilizan en situaciones donde es necesario indicar si algo está seleccionado o no. Aparte de los estados anteriores, se puede indicar un tercer estado de indeterminación. Este tipo de botones puede aparecer en widgets RowColumn, incluyendo los `RadioBox` y `CheckBox`. Los primeros son conjuntos de togglebuttons donde solo uno puede estar seleccionado, en este caso el estado del botón viene dado por un pixmap en forma de diamante. En el CheckBox puede aparecer más de uno seleccionado y el pixmap que indica el estado del botón es un cuadrado. En ambos casos el comportamiento viene fijado por el widget padre, un widget RowColumn. Este tipo de botones tienen las siguientes características:

1. Posee callbacks cuando es armado y desarmado. Cuando cambia de estado tiene un callback asociado que es `XmNvalueChangedCallback`.

2. Mediante el recurso `XmNindicatorType` se indica si uno o más botones pueden estar activados a la vez.

3. El recurso `XmNindicatorOn` especifica que tipo de pixmap es dibujado junto a la etiqueta del botón para indicar la selección.

4. Con `XmNvisibleWhenOff` se puede hacer que cuando el botón no está seleccionado no aparezca.

En la parte inferior de la figura 3.20 se pueden ver dos conjuntos de togglebutton, el de la derecha como `RadioBox` y el de la izquierda como `CheckBox`. La creación de un RadioBox se puede hacer de dos formas. Una es mediante la rutina `XmCreateRadioBox` y otra es como aparece en el siguiente código perteneciente a `botones.c`.

```
n=0;
XtSetArg(args[n],XmNisHomogeneous,True); n++;
XtSetArg(args[n],XmNpacking,XmPACK_COLUMN); n++;
XtSetArg(args[n],XmNradioBehavior,True); n++;
XtSetArg(args[n],XmNrowColumnType,XmWORK_AREA); n++;
radio=XmCreateRowColumn(formulario,"radio",args,n);
XtManageChild(radio);

n=0;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Toggle Button 1",
        char_set)); n++;
boton1=XmCreateToggleButton(radio,"boton",args,n);
XtManageChild(boton1);
```

Por otra parte, la creación de un CheckBox es un widget RowColumn cuyos widgets hijo son botones del tipo toggle, como se puede ver a continuación.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
n=0;
check=XmCreateRowColumn(formulario,"radio",args,n);
XtManageChild(check);
n=0;
XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Toggle Button 1",
        char_set)); n++;
boton1=XmCreateToggleButton(check,"boton",args,n);
XtManageChild(boton1);
```

CascadeButton

Un botón de tipo cascada se utiliza dentro de los menús para desplegar un menú desplegable cuando se activa. Para indicar que existe un menú desplegable asociado a un botón de este tipo, aparece al lado del mismo un pixmap (normalmente una flecha). Este menú desplegable se le asocia al botón mediante el recurso `XmNsubMenuId`. En cuanto a los callbacks, existen dos que son `XmNactivateCallback` y `XmNcascadingCallback`. El primero se produce cuando se activa el botón, mientras que el segundo se activa antes de desplegarse el menú asociado. El ejemplo de este tipo de botones se verá en el apartado dedicado a los menús.

3.8.3 Listas

Una lista (`XmList`) es un widget que contiene una vector de elementos (texto) de los cuales el usuario puede elegir uno o mas. Cada elemento corresponde a un XmString. Este widget tiene cuatro posibilidades de selección, definidas por el valor asignado al recurso `XmNselectionPolicy` que puede ser:

- `XmSINGLE_SELECT` En este modo solo se puede seleccionar un único elemento de la lista. Cuando se selecciona un elemento en este modo se desactiva la selección de otro elemento si la hubiera.

- `XmBROWSE_SELECT` Este modo es similar al anterior, con la diferencia que la selección del único elemento se puede hacer mediante el arrastre del raton por los elementos.

- `XmMULTIPLE_SELECT` En este modo a diferencia de los anteriores permite la seleccion de varios elementos. Asi cuando se selecciona un nuevo elemento los ya seleccionados no se desactivan.

- `XmEXTENDED_SELECT` En este modo también se pueden seleccionar varios elementos de la lista, consecutivos o no.

Cuando se realiza una seleccion por parte del usuario se llama a alguno de los siguiente callbacks, dependiendo del tipo de seleccion utilizada: `XmNsingleSelectionCallback`, `XmNbrowseSelectionCallback`, `XmNmultipleSelectionCallback` o `XmNextendedSelectionCallback`. Aparte de los anteriores callbacks existe otro que se activa siempre que se realiza un doble *click* del raton en un elemento. Este callback es `XmNdefaultActionCallback`.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Aparte del recurso que define el tipo de selección hay otros que indican el numero de elementos que se pueden mostrar a la vez en la lista (`XmNvisibleItemCount`), la dirección de los elementos (`XmNstringDirection`) y que acción se toma cuando un elemento es mayor que el ancho de la lista (`XmNlistSizePolicy`). Además existen una serie de rutinas que permiten realizar una serie de operaciones sobre las listas. Algunas de estas rutinas son las siguientes. Una listado completo se puede obtener en la página de manual de XmList.

- `XmListAddItem` Añade un elemento a la lista en un posición dada.
- `XmListDeleteItem` Elimina un elemento de la lista que coincide con el XmString que se le pasa a la rutina.
- `XmListItemPos` Devuelve la primera posición en la lista, donde aparece el elemento que se indica.
- `XmListDeselectAllItems` Elimina todas las selecciones que pudieran estar hechas sobre la lista.

3.8.4 Escalas

El widget escala (`XmScale`) permite seleccionar un valor comprendido entre dos extremos, mediante el desplazamiento de una barra deslizante. El funcionamiento es similar a las barras deslizantes, con la diferencia de que en este caso se muestra el valor. Aunque este widget no pertenece a la clase `XmPrimitive` ya que acepta hijos, se ha incluido en esta sección porque es un elemento de interfaz con el usuario. Una demostración de este widget se puede ver en la figura 3.21, que corresponde a la ejecución del programa `escala.c`.



Figura 3.21: Widget Scale

El valor mínimo y máximo de la escala se establecen mediante los recursos `XmNminimum` y `XmNmaximum`. También se puede indicar si el máximo aparece al comienzo o al final por medio del recurso `XmNprocessingDirection`. El valor seleccionado a medida que se desliza la barra se va a incrementar en uno. Sin embargo puede modificarse para que el incremento sea diferente mediante el recurso `XmNscaleMultiple`. Los callbacks que este

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

widget tiene asociados son: XmNvaluedChangedCallback y XmNdragCallback. El primero se activa cada vez que la selección cambia de valor, pero sin arrastrar la barra. El segundo se llama cuando se cambia el valor mediante el arrastre de la barra. El widget puede estar en posición horizontal o vertical, según el valor del recurso XmNorientation. La creación de un widget escala y la asignación de algunos de los recursos antes mencionados se puede ver a continuación:

```
n=0;
XtSetArg(args[n],XmNshowValue,True); n++;
XtSetArg(args[n],XmNtitleString,xmcad); n++;
XtSetArg(args[n],XmNorientation,XmHORIZONTAL); n++;
XtSetArg(args[n],XmNmaximum,100); n++;
escala=XmCreateScale(marco,"escala",args,n);
XtManageChild(escala);
XtAddCallback(escala,XmNvalueChangedCallback,maneja_escala,NULL);
XtAddCallback(escala,XmNdragCallback,maneja_escala,NULL);
```

Si se quiere que aparezcan textos que indiquen los valores intermedios se puede hacer mediante la creación de etiquetas que van a ser hijos del widget escala.

```
for (i=0;i<=10;i++) {
  n=0;
  sprintf(valor_cad,"%d",i*10);
  etiqueta[i]=XmCreateLabel(escala,valor_cad,args,n);
}
XtManageChildren(etiqueta,i);
```

El código del callback que se ha asociado a los eventos es el siguiente. En este ejemplo se puede ver el uso del campo reason para diferenciar el tipo de evento que ha producido el callback.

```
void maneja_escala(Widget w, XtPointer client_data, XtPointer call_data)
{
  int valor,razon;
  XmScaleCallbackStruct *call_value=(XmScaleCallbackStruct *)call_data;
  razon=call_value->reason;
  valor=call_value->value;
  switch (razon) {
  case XmCR_VALUE_CHANGED:
    printf("Razon: XmCR_VALUE_CHANGED, valor: %d\n",valor);
    break;
  case XmCR_DRAG:
    printf("Razon: XmCR_DRAG, valor: %d\n",valor);
    break;
  }
}
```

3.8.5 Textos

Los widgets de textos (XmText y XmTextField) permiten la posibilidad de edición de texto en una sola línea o en múltiples. El funcionamiento de ambos es muy similar, excepto en que el XmTextField solo permite una línea, por tanto el widget Text dispone de algunas rutinas que no tiene el XmTextField. Un elemento a tener en cuenta es que el

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

widget XmText puede estar limitado a una sola línea, aunque es más eficiente el uso del XmTextField en estos casos.

En la figura 3.22 se muestra una ventana que contiene dos widgets de textos. El primero corresponde a un XmTextField mientras que el segundo a un XmText.

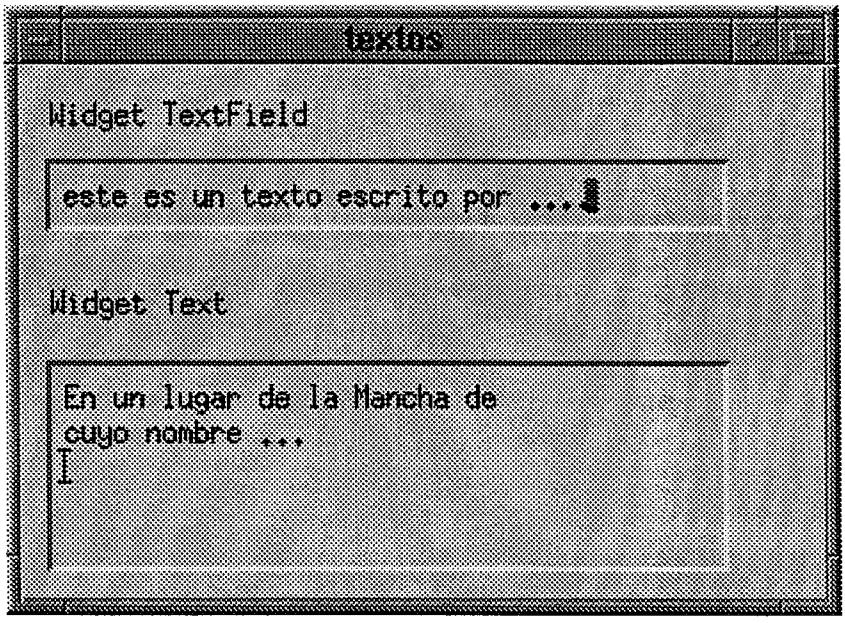

Figura 3.22: Widgets de texto en Motif

El código que crea y maneja el widget XmTextField que aparece en la figura 3.22 es el siguiente:

```
n=0;
XtSetArg(args[n],XmNx,10); n++;
XtSetArg(args[n],XmNy,35); n++;
XtSetArg(args[n],XmNcolumns,40); n++;
texto1=XmCreateTextField(tablon,"texto",args,n);
XtManageChild(texto1);
XtAddCallback(texto1,XmNactivateCallback,maneja_texto1,NULL);
```

Los recursos que más se utilizan con este widget se comentan a continuación. El ancho en columnas (cada columna corresponde a un carácter) viene dado por el recurso `XmNcolumns`. El recurso anterior no limita el tamaño del texto que se puede escribir. Para limitar el tamaño del texto se utiliza el recurso `XmNmaxLength`. El recurso `XmNeditable` indica si el widget se puede editar por parte del usuario o no. El texto que va a aparecer en el widget está contenido en el recurso `XmNvalue`, y se corresponde con una cadena de caracteres. Este recurso se puede utilizar tanto para poner un texto cuando se crea el widget, como para leer el texto que ha escrito el usuario. Otro recurso utilizado es el callback `XmNactivateCallback`. A continuación se muestra el código correspondiente al anterior callback, donde se lee el texto introducido y se imprime en la terminal.

```
void maneja_texto1(Widget w, XtPointer client_data, XtPointer call_data)
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
{
  Arg args[16];
  register int n;
  char *cadena;

  n=0;
  XtSetArg(args[n],XmNvalue,&cadena); n++;
  XtGetValues(w,args,n);

  printf("Texto escrito: %s\n",cadena);
}
```

Aparte de los recursos que posee el widget, existen una serie de rutinas para el manejo del texto en un widget TextField. Todas estas rutinas aparecen al final de la página de manual del `XmTextField`. Algunas de ellas son:

- `XmTextFieldGetInsertionPosition` Obtiene la posición del cursor.
- `XmTextFieldInsert` Inserta una cadena de caracteres en un widget TextField a partir de una determinada posición.
- `XmTextFieldReplace` Reemplaza entre dos posiciones el texto contenido en un widgetTextField con una cadena de caracteres.
- `XmTextFieldSetEditable` Activa o desactiva la posibilidad de editar un TextField.
- `XmTextFieldSetInsertionPosition` Pone la posición del cursor a una determinada posición.

El widget Text que aparece en la figura 3.22 se ha creado con el siguiente código:

```
n=0;
XtSetArg(args[n],XmNx,10); n++;
XtSetArg(args[n],XmNy,110); n++;
XtSetArg(args[n],XmNcolumns,40); n++;
XtSetArg(args[n],XmNrows,5); n++;
XtSetArg(args[n],XmNeditMode,XmMULTI_LINE_EDIT); n++;
texto2=XmCreateText(tablon,"texto",args,n);
XtManageChild(texto2);
XtAddCallback(texto2,XmNvalueChangedCallback,maneja_texto2,NULL);
```

Entre los recursos del este widget Text se encuentran el `XmNcolumns` que tiene la misma función que en el TextField. Además, si se van a tener varias líneas se debe especificar mediante el recurso `XmNeditMode`, asignándole el valor `XmMULTI_LINE_EDIT`. El número de líneas que va a tener el widget se indica con el recurso `XmNrows`. Aparte de los anteriores recursos también están los recursos `XmNmaxLength`, `XmNvalue` y `XmNeditable`, que funcionan igual que en el TextField. Un callback que se utiliza con este tipo de widget es el `XmNvalueChangedCallback` que se activa después de cada modificación que se hace en el texto. Al igual que en el widget TextField, para este widget existen también una serie de rutinas para manejar el texto que contiene. Algunas de estas rutinas son las siguientes:

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

- `XmTextFindString` Busca una cadena de caracteres en el texto que contiene el widget, a partir de una posición especificada y en la dirección dada. Devuelve la primera posición donde aparece la cadana buscada.

- `XmTextGetInsertionPosition` Igual que en el widget TextField.

- `XmTextGetString` Devuelve la cadena de caracteres que se existe en el widget.

- `XmTextInsert` Igual que en el widget TextField.

- `XmTextSetEditable` Igual que en el widget TextField.

- `XmTextSetInsertionPosition` Igual que en el widget TextField.

- `XmTextSetString` Establece la cadena de caracteres que se pasa como texto del widget.

- `XmTextXYToPos` Devuelve la posición del caracter más cercano a las coordenadas X e Y dentro del widget.

Una opción del widget Text es que muestre unas barras de desplazamiento. Para conseguir esto se puede utilizar la rutina `XmCreateScrolledText`.

3.9 Menú y Opciones

3.9.1 Menús

Los menú son probablemente la forma más común de presentar al usuario opciones y comandos. En *OSF/Motif* se definen cuatro tipos de menú: *MenuBar*, *PopupMenu*, *PulldownMenu* y *OptionMenu*.

- `MenuBar` Este menú normalmente consiste de una línea de botones del tipo cascada. Cuando el usuario pulsa alguno de ellos, se despliega un menú pulldown. Este tipo de menú suele estar manejado por algún elemento de la aplicación, como puede ser un widget del tipo MainWindow.

- `PopupMenu` Este menú contiene una serie de opciones que se aplican a algún elemento de la aplicación. Este menú no esta visible hasta que el usuario lo activa mediante el botón derecho del ratón, cuando se encuentra en la ventana que tiene asociado el menú.

- `PulldownMenu` Este menú se asocia a un botón de tipo cascada dentro de un menú MenuBar, un Popup o incluso dentro de otro menú Pulldown. El menú no es visible hasta que el usuario pulsa el botón de tipo cascada.

- `OptionMenu` Este menú lo que permite es elegir una opción entre un conjunto, dejando visible la opción elegida.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

A continuación se describen algunos de los widgets que intervienen en la creación de un menú. Para cada menú Pulldown o Popup se necesita un widget `XmMenuShell` ya descrito en la sección 3.6. Estos shell son ignorados por el manejador de ventanas, ya que son ventanas temporales que exigen la atención del usuario antes de realizar cualquier otra acción. El widget sobre el que se crean todos los menús es el `XmRowColumn` (sección 3.7.4), debido a la distribución en columna de los widgets hijos de un menú. Por ultimo, están los elementos que informan al usuario o sobre los que éste actúa, como pueden ser etiquetas y botones. Dentro de los botones, un tipo bastante importante son los botones Cascade.

Las rutinas que proporciona *OSF/Motif* para la creación de los menús son las siguientes: `XmCreateMenBar`, `XmCreatePulldownMenu`, `XmCreatePopupMenu` y `XmCreate OptionMenu`. Estas rutinas crean un widget del tipo `XmRowColumn` con el recurso `XmNrow ColumnType` a los siguientes valores respectivamente: `XmMENU_BAR`, `XmPULLDOWN_MENU`, `XmPOPUP_MENU` y `XmOPTION_MENU`.

Antes de entrar en las técnicas para crear los distintos tipos menús, se va a indicar que tipos de widgets hijos va a permitir cada tipo de menú.

	MenuBar	Popup	Pulldown	Option
Boton Cascade	X	X	X	X
Boton PushButton		X	X	X
Boton Toggle		X	X	
Etiqueta		X	X	X
Separador		X	X	X

Creación de un menú MenuBar

Al igual que el resto de los menús, el MenuBar se crea a partir de un widget RowColumn. Pero como se vió anteriormente en *OSF/Motif* existe una rutina para la creación de este tipo de widget (`XmCreateMenuBar`). En el programa `menus.c`, que es una modificación del `ventana-pral.c`, se muestra como se pueden crear los distintos tipos de menú (figura 3.23). La parte de este programa donde se crea el MenuBar es:

```
n=0;
menub=XmCreateMenuBar(padre,"menu",args,n);
XtManageChild(menub);
```

Debido a que este tipo de menú solo puede tener como hijos a botones de tipo Cascade, de él solo pueden colgar menús Pulldown. A continuación se explican los pasos necesarios para crear y asociar estos menús un MenuBar.

1. Se crea el menú Pulldown como hijo del MenuBar mediante la rutina `XmCreate PulldownMenu`.

2. Se crean y añaden los botones, etiquetas y separadores que va a contener el menú Pulldown.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

3. Por ultimo se crea y maneja un botón Cascade como hijo del MenuBar, asociando al recurso `XmNsubMenuId` del botón, el identificador del menú creado en el paso 1.

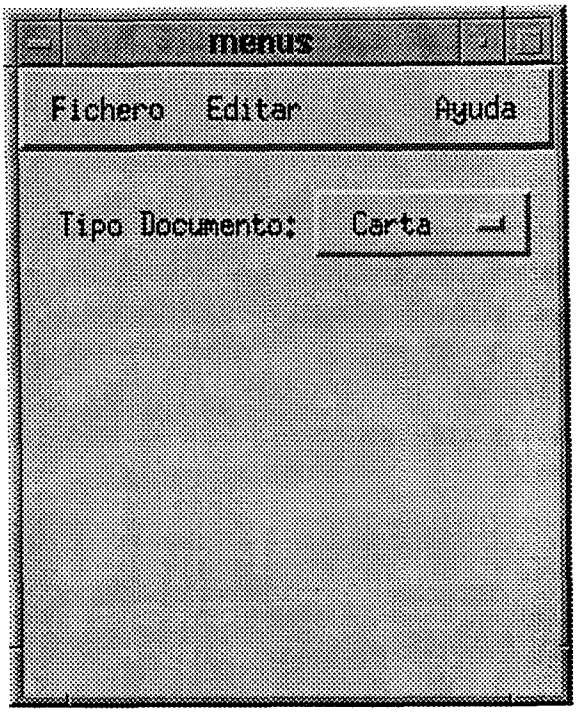

Figura 3.23: Distintos tipos de menu

Las siguientes líneas corresponden a la creación de un menú Pulldown en el programa `menus.c`.

```
n=0;
menupane=XmCreatePulldownMenu(menub,"menu",args,n);

n=0;
boton=XmCreatePushButton(menupane,"Abrir",args,n);
XtManageChild(boton);

n=0;
boton=XmCreatePushButton(menupane,"Guardar",args,n);
XtManageChild(boton);

n=0;
boton=XmCreatePushButton(menupane,"Salir",args,n);
XtManageChild(boton);

XtAddCallback(boton,XmNactivateCallback,salir,NULL);

n=0;
XtSetArg(args[n],XmNsubMenuId,menupane); n++;
cascada=XmCreateCascadeButton(menub,"Fichero",args,n);
XtManageChild(cascada);
```

Un elemento que aparece con frecuencia en los MenuBar es el botón de ayuda. Según la *Guía de estilo de OSF/Motif* este botón debe ir en el lado derecho de la barra de menú. En el MenuBar se ha definido para el botón de ayuda un recurso denominado

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

XmNmenuHelpWidget, al que se le asigna el identificador del botón Cascade que va a contener la ayuda. En el programa ventana-pral.c se utilizo este botón, y se creó de la siguiente forma:

```
n=0;
XtSetArg(args[n],XmNsubMenuId,menupane); n++;
cascada=XmCreateCascadeButton(menub,"Ayuda",args,n);
XtManageChild(cascada);

n=0;
XtSetArg(args[n],XmNmenuHelpWidget,cascada); n++;
XtSetValues(menub,args,n);
```

Creación de un menú Popup

Igual que el menú Pulldown, el menú Popup es un widget RowColumn y también existe una rutina para crearlo que es XmCreatePopupMenu. Con esta rutina se crea un widget Shell y dentro del mismo un RowColumn. Una vez creado el menú, se añaden los botones, etiquetas o separadores que el usuario estime necesario. A continuación se muestra como se crea un menú Popup, dentro del programa menus.c.

```
n=0;
menupane=XmCreatePopupMenu(padre,"menu",args,n);

n=0;
etiqueta=XmCreateLabel(menupane,"Opciones",args,n);
XtManageChild(etiqueta);

n=0;
separador=XmCreateSeparator(menupane,"",args,n);
XtManageChild(separador);

...
```

A diferencia del menú Pulldown, que necesitaba de un botón del tipo Cascade para aparecer. En este tipo de menú, es responsabilidad de la aplicación establecer el mecanismo para la aparición del menú Popup (figura 3.24). La forma más común es mediante la adición de un manejador de eventos para el evento que causa la aparición del menú. En el programa menus.c se ha optado por utilizar el botón derecho para tal tarea, y por tanto es necesario añadir un manejador de eventos. El código del mismo es:

```
void sacar_popup(Widget w, XtPointer client_data, XEvent *event)
{
  Widget menu_popup=(Widget)client_data;
  XButtonEvent *evento=(XButtonEvent *)event;

  if (evento->button == Button3) {
    XmMenuPosition(menu_popup,evento);
    XtManageChild(menu_popup);
  }
}
```

y la asociación de la anterior rutina al evento del botón derecho del ratón, es:

```
XtAddEventHandler(tablon,ButtonPressMask,False,(XtEventHandler)sacar_popup,
                  (XtPointer)menu_popup);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

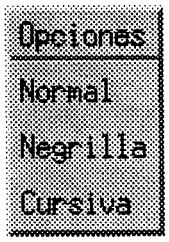

Figura 3.24: Menu Popup

Creación de un menú Option

La distribución de un menú de opciones es diferente a las vistas hasta ahora para los otros tipos. Un menú Option consta de un widget RowColumn que contiene una etiqueta y un botón cascada, al que se le asocia un menú Pulldown. Como puede verse en la figura 3.23 una pequeña barra horizontal distingue este tipo de menú de los otros. En el botón de cascada se muestra como etiqueta la selección actual. A continuación se muestra la parte del código del programa `menus.c` correspondiente a la creación del menú de opciones que aparece en la ventana.

```
n=0;
XtSetArg(args[n],XmNtearOffModel,XmTEAR_OFF_ENABLED); n++;
pulldown=XmCreatePulldownMenu(padre,"",args,n);

n=0;
etiqueta=XmCreateLabel(pulldown,"Opciones",args,n);
XtManageChild(etiqueta);

n=0;
separador=XmCreateSeparator(pulldown,"",args,n);
XtManageChild(separador);

n=0;
boton=XmCreatePushButton(pulldown,"Carta",args,n);
XtManageChild(boton);

n=0;
boton=XmCreatePushButton(pulldown,"Articulo",args,n);
XtManageChild(boton);

n=0;
boton=XmCreatePushButton(pulldown,"Libro",args,n);
XtManageChild(boton);

xmcad=XmStringCreateLtoR("Tipo Documento:",char_set);
n=0;
XtSetArg(args[n],XmNlabelString,xmcad); n++;
XtSetArg(args[n],XmNsubMenuId,pulldown); n++;
menupane=XmCreateOptionMenu(padre,"menu_option",args,n);
XtManageChild(menupane);
```

En el código anterior, el widget RowColumn que va a contener la etiqueta y el botón cascada se crea con la rutina `XmCreateOptionMenu`, y la asignación del widget etiqueta y menú pulldown se hace mediante los recursos `XmNlabelString` y `XmNsubMenuId` respectivamente. Indicar que el menú pulldown en este caso va a tener asociado al recurso `XmNtearOffModel` el valor `XmTEAR_OFF_ENABLED`.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

3.9.2 Selección Combinada

Este tipo de widget al igual que el siguiente no son widget del tipo menú, pero como permiten la selección de una determinada opción entre varias se ha incluido en esta sección. El widget `XmComboBox` consta de una lista y un campo de texto, que permite que la selección sea realizada por el usuario de dos formas: eligiendo una opción de la lista o escribiendo la opción deseada. Por defecto se muestran la lista y el campo de texto, pero se puede modificar su comportamiento por medio del recurso `XmNcomboBoxType` según los valores de la siguiente tabla.

Valor	Campo de datos	Lista
`XmCOMBO_BOX`	editable	se muestra
`XmDROP_DOWN_COMBO_BOX`	editable	oculta
`XmDROP_DOWN_LIST`	no editable	oculta

En *OSF/Motif* existen tres tipos de rutinas para crear este widget: `XmCreate ComboBox`, `XmCreateDropDownComboBox` y `XmCreateDropDownList`. Esta rutinas crean un ComboBox de los tipos que se vieron en la tabla anterior, respectivamente. En el vector de argumentos que se pasa a las rutinas, pueden asignarse valores a recursos propios de algunos de los widgets que compone el ComboBox. Por ejemplo, se pueden asignar valores a los recursos `XmNitems` o `XmNvisibleItemCount` del widget lista. Para obtener el identificador de alguno de los widgets se puede utilizar la rutina `XmNameToWidget` con `*List` o `*Text` como uno de sus parámetros.

Cuando el widget tiene una lista, aparece una flecha que cuando se pulsa se despliega. El aspecto de esta flecha viene dado por los siguientes recursos `XmNlayoutDirection`, `XmNarrowSize` y `XmNarrowSpacing`. Un callback que esta asociado a este widget es `XmNselectionCallback`, que se activa cada vez que se realiza una selección. En la figura 3.25 se muestra la salida del programa `combo.c`. A continuación se muestra un trozo del código.

```
n=0;
XtSetArg(args[n],XmNwidth,150); n++;
XtSetArg(args[n],XmNcomboBoxType,XmDROP_DOWN_COMBO_BOX); n++;
XtSetArg(args[n],XmNarrowSpacing,5); n++;
XtSetArg(args[n],XmNitems,xmcad); n++;
XtSetArg(args[n],XmNitemCount,NUM_ITEMS); n++;
XtSetArg(args[n],XmNvisibleItemCount,4); n++;
combo=XmCreateComboBox(tablon,"combo",args,n);
XtManageChild(combo);
XtAddCallback(combo,XmNselectionCallback,maneja_seleccion,NULL);
```

3.9.3 Selección Rotatoria

Este tipo de widget (`XmSpinBox`) permite al usuario elegir una opción entre un conjunto mútuamente exclusivas. Para ello se muestran dos flechas que hacen que las opciones vayan mostrándose en uno de los dos sentidos. Cuando se llega a uno de los finales, se vuelve a mostrar la primera empezando por el otro extremo. Si alguna de las flechas se mantiene pulsada, las opciones van cambiando hasta que se suelte. Los hijos del

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

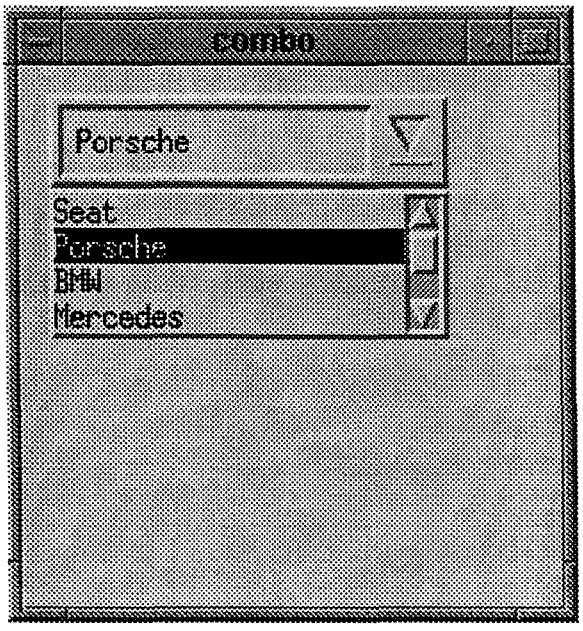

Figura 3.25: Salida del programa `combo.c`

XmSpinBox pueden ser texto, etiquetas o separadores. El texto solo puede ser de dos tipos: numérico o cadena de caracteres. Para especificar el tipo del hijo se utiliza el recurso `XmNspinBoxChildType`, que es heredado por los hijos. En la figura 3.26 se muestra una ventana que contiene un widget XmSpinBox con dos widgets hijos no numéricos. Para que la flechas actúen sobre alguno de los campos, se debe haber activado anteriormente, pulsando un botón del ratón en su interior.

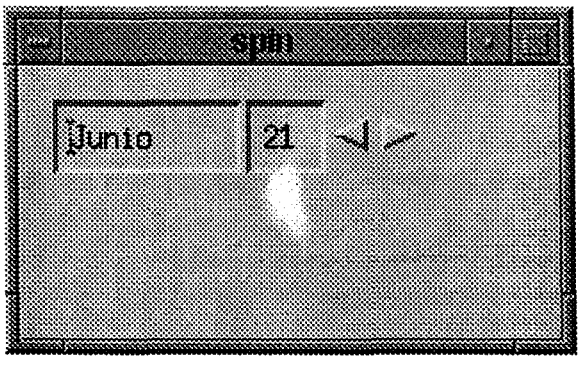

Figura 3.26: Widget SpinBox

Los callbacks que están asociados a este widget son: `XmNmodifyVerifyCallback` y `XmNvalueChangedCallback`. El primero se activa siempre que una de las flechas es pulsada. El segundo se activa siempre que se pulsa una flecha y la posición del SpinBox cambia. Algunos recursos que posee este widget son:

- `XmNvalues` Vector de XmString que contiene los elementos del widget hijo, cuando éste es tipo cadena de caracteres.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

- `XmNnumValues` Indica el numero de elementos que contiene el vector de XmString asignado al anterior recurso.

- `XmNminimumValue` Valor mínimo cuando el widget hijo contiene valores numéricos.

- `XmNmaximumValue` Valor máximo cuando el widget hijo contiene valores numéricos.

- `XmNincrementValue` Valor en que se incrementa/decrementa el valor numérico.

- `XmNdecimalPoints` Numero de posiciones decimales que tiene los valores numéricos.

- `XmNposition` Da la posición actual en el rango de números válidos o en el vector de XmString.

Un ejemplo de creación de este tipo de widget se puede ver en las siguientes líneas de código, correspondientes al programa `spin.c`.

```
n=0;
XtSetArg(args[n],XmNx,10); n++;
XtSetArg(args[n],XmNy,10); n++;
rotativo1=XmCreateSpinBox(tablon,"rotativo",args,n);

crea_meses();

n=0;
XtSetArg(args[n],XmNwidth,70); n++;
XtSetArg(args[n],XmNvalues,meses); n++;
XtSetArg(args[n],XmNnumValues,12); n++;
XtSetArg(args[n],XmNselectionPolicy,XmSINGLE_SELECT); n++;
XtSetArg(args[n],XmNeditable,False); n++;
texto_rot1=XmCreateTextField(rotativo1,"texto_rot1",args,n);

crea_dias();

n=0;
XtSetArg(args[n],XmNwidth,30); n++;
XtSetArg(args[n],XmNvalues,dias); n++;
XtSetArg(args[n],XmNnumValues,31); n++;
XtSetArg(args[n],XmNselectionPolicy,XmSINGLE_SELECT); n++;
XtSetArg(args[n],XmNeditable,False); n++;
texto_rot2=XmCreateTextField(rotativo1,"texto_rot2",args,n);

XtManageChild(rotativo1);

XtAddCallback(rotativo1,XmNmodifyVerifyCallback,maneja_cambio,NULL);

XtManageChild(texto_rot1);
XtManageChild(texto_rot2);
```

En el código anterior, además de la creación del widget SpinBox, se puede ver la creación de los widget de textos que van a ser los hijos del SpinBox. Un aspecto importante a tener en cuenta con este tipo de widget es el orden en que se manejan los hijos y el XmSpinBox.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

3.10 Cuadros de diálogo

Este tipo de widgets proporciona un medio por el cual se establece una comunicación entre el usuario de una aplicación y la aplicación propiamente dicha. Un cuadro de diálogo normalmente realiza una pregunta al usuario o presenta algún tipo de información, de forma que la aplicación se puede detener hasta que el usuario responda. Esta forma de funcionar es a lo que se denomina *modal*. Por otro lado si se puede hacer que la aplicación continúe sin que el usuario responda al cuadro de diálogo, entonces se denomina *modeless*.

3.10.1 ErrorDialog, InformationDialog, WarningDialog, WorkingDialog y QuestionDialog

Estos widget consisten en un DialogShell, que tiene como hijo un widget del tipo `XmMessageBox`. El XmMessageBox posee una serie de recursos como un mensaje que se muestra en la mitad superior de la ventana, un separador y tres botones de tipo PushButton. La diferencia entre los cuadros de estos diálogos, radica en que junto al mensaje aparece un símbolo que es diferente en cada uno. Los símbolos son los que aparecen en la siguiente tabla, junto con la rutina que crea cada widget.

Cuadro	Símbolo	Rutina
ErrorDialog	Signo de prohibición	XmCreateErrorDialog
InformationDialog	Letra 'i'	XmCreateInformationDialog
WarningDialog	Signo de exclamación	XmCreateWarningDialog
WorkingDialog	Reloj de arena	XmCreateWorkingDialog
QuestionDialog	Signo de interrogación	XmCreateQuestionDialog

Los recursos que poseen estos widgets son los del XmMessageBox. El recurso que define el mensaje a mostrar es `XmNmessageString`, al cual se le asigna una XmString. Los tres botones que aparecen tienen por defecto las etiquetas `Ok`, `Cancel` y `Help`. Para cambiar las etiquetas anteriores se utilizan los siguentes recursos: `XmNokLabelString`, `XmNcancelLabelString` y `XmNhelpLabelString` respectivamente. De la misma forma cada botón tiene asociado un callback de activación: `XmNokCallback`, `XmNcancelCallback` y `XmNhelplCallback`.

En las figuras 3.27, 3.28, 3.29, 3.30 y 3.31, se muestran los distintos cuadros de diálogo que se han visto en este apartado. En el programa `dialogos.c` se encuentra el código que implementa los ejemplos de las anteriores figuras. Como ejemplo, la parte donde se crea el ErrorDialog es la siguiente:

```
xmcad=XmStringCreateLtoR("Esto es un error que se ha producido",char_set);
n=0;
XtSetArg(args[n],XmNmessageString,xmcad); n++;
dialogo=XmCreateErrorDialog(toplevel,"",args,n);
XtManageChild(dialogo);
XmStringFree(xmcad);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

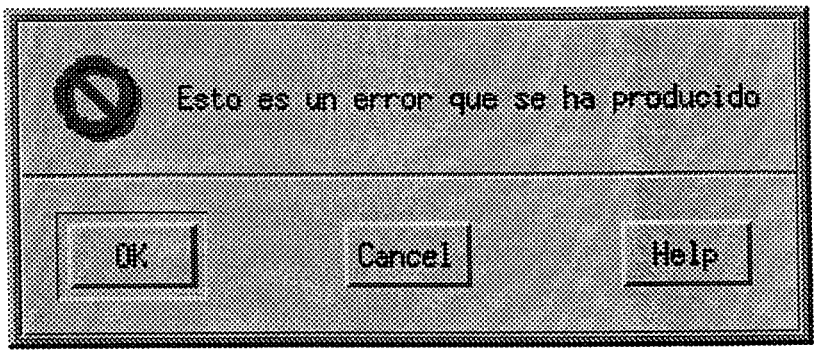

Figura 3.27: ErrorDialog

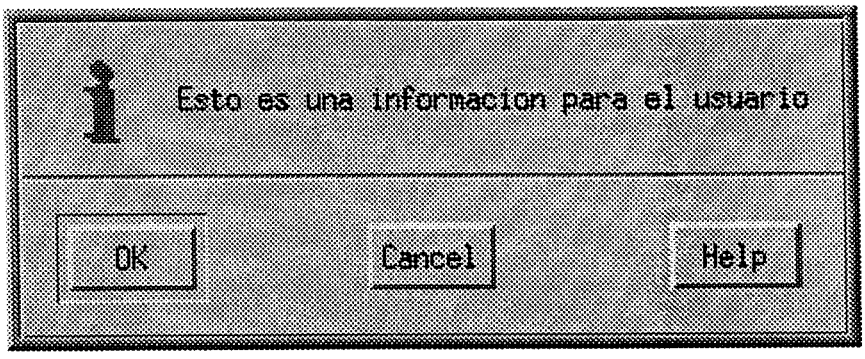

Figura 3.28: InformationDialog

3.10.2 FileSelectionDialog

Este cuadro de diálogo se utiliza cuando el usuario tiene que seleccionar un fichero dentro de un directorio. Este widget pertenece a la clase `XmFileSelectionBox` y la rutina para crearlo es `XmCreateFileSelectionBox`. El aspecto de este cuadro de diálogo se puede ver en la figura 3.32. Para la selección del fichero se puede especificar un filtro así como el directorio a mostrar. Estas opciones se dan mediante los recursos `XmNpattern` y `XmNdirectory` respectivamente, asignándoles una XmString. La parte del programa `dialogos.c` donde se crea el FileSelection se puede ver a continuación.

```
camino=XmStringCreateLtoR("/tmp",char_set);
mascara=XmStringCreateLtoR("*",char_set);

n=0;
XtSetArg(args[n],XmNdirectory,camino); n++;
XtSetArg(args[n],XmNpattern,mascara); n++;
dialogo=XmCreateFileSelectionDialog(toplevel,"",args,n);
XtManageChild(dialogo);

XtAddCallback(dialogo,XmNokCallback,maneja_ok,(XtPointer)1);

XmStringFree(camino);
XmStringFree(mascara);
```

El resultado de la selección se encuentra en el campo `value` de la estructura `XmSelectionBoxCallbackStruct`, que se le pasa a los callbacks asociados a los cuatro

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

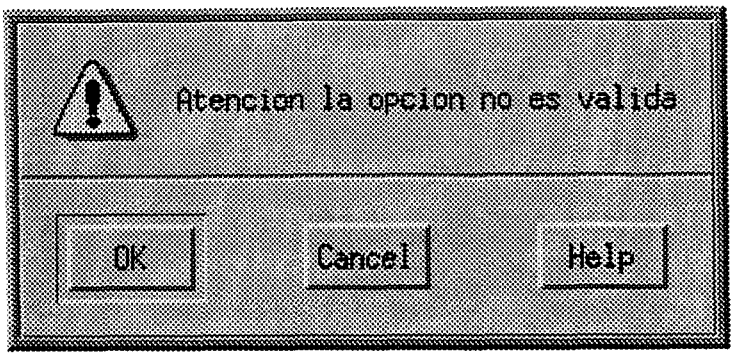

Figura 3.29: WarningDialog



Figura 3.30: WorkingDialog

botones que posee el cuadro de diálogo. De los cuatro botones, tres son iguales a los vistos en el apartado anterior y con los mismos recursos asociados. El botón que aparece nuevo en este cuadro de diálogo es el de `Filter` y posee los recursos `XmNfilterLabelString` y `XmNfilterCallback` para la etiqueta y el callback, respectivamente.

3.10.3 PromptDialog

Este widget se utiliza para solicitar al usuario de la aplicación que introduzca algún dato. Pertenece a la clase `XmSelectionBox` por lo que el manejo de los botones es igual que en el FileSelectionDialog. El aspecto que tiene este widget se puede ver en la figura 3.33, y el código que lo crea dentro del programa `dialogos.c` es el siguiente.

```
xmcad=XmStringCreateLtoR("Teclee su nombre",char_set);
n=0;
XtSetArg(args[n],XmNselectionLabelString,xmcad); n++;
dialogo=XmCreatePromptDialog(toplevel,"",args,n);
XtManageChild(dialogo);
XtAddCallback(dialogo,XmNokCallback,maneja_ok,(XtPointer)3);
XmStringFree(xmcad);
```

Cuando se pulsa alguno de los tres botones (iguales a los del cuadro de diálogo ErrorDialog), lo introducido por el usuario se encuentra en el campo `value` de la estructura `XmSelectionBoxCallbackStruct`.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005



Figura 3.31: QuestionDialog

3.10.4 SelectionDialog

Este cuadro de diálogo muestra al usuario una serie de opciones en una lista para que seleccione una de ellas. La opción seleccionada se muestra en un cuadro de texto. Además posee cuatro botones similares a los del FileSelectionDialog, como se puede ver en la figura 3.34, pero en lugar del botón `Filter` en este widget aparece como `Apply`.

```
switch (opcion)\\
case 1:\\
  printf("Fichero Selecionado: %s\n",cadena);\\
  XtDestroyWidget(w);\\
  break;\\
case 2:\\
  printf("Widget Selecionado: %s\n",cadena);\\
  break;\\
case 3:\\
  printf("Su nombre es: %s\n",cadena);\\
```

Los recursos que controlan la lista de opciones son `XmNlistItems`, `XmNlistItem Count` y `XmNlistVisibleItemCount`, que van a contener un vector de XmString con las opciones, el número de elementos del vector, y por último cuantas opciones van a estar visibles. En el supuesto de que el vector tenga más opciones de las visibles se crean unas barras deslizantes, de la misma forma que ocurría con el widget List. Sobre la lista de opciones puede aparecer una etiqueta, mediante la asignación al recurso `XmNlistLabelString` una XmString. La ventana de selección también puede tener una etiqueta haciendo uso del recurso `XmNselectionLabelString`.

```
for (n=0;n<NUM_VALORES;n++)
  valores[n]=XmStringCreateLtoR(cad_valores[n],char_set);
xmcad=XmStringCreateLtoR("Widgets",char_set);
camino=XmStringCreateLtoR("Widget Seleccionado",char_set);

n=0;
XtSetArg(args[n],XmNlistItems,valores); n++;
XtSetArg(args[n],XmNlistItemCount,NUM_VALORES); n++;
XtSetArg(args[n],XmNlistVisibleItemCount,5); n++;
XtSetArg(args[n],XmNlistLabelString,xmcad); n++;
XtSetArg(args[n],XmNselectionLabelString,camino); n++;
dialogo=XmCreateSelectionDialog(toplevel,"",args,n);
XtManageChild(dialogo);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

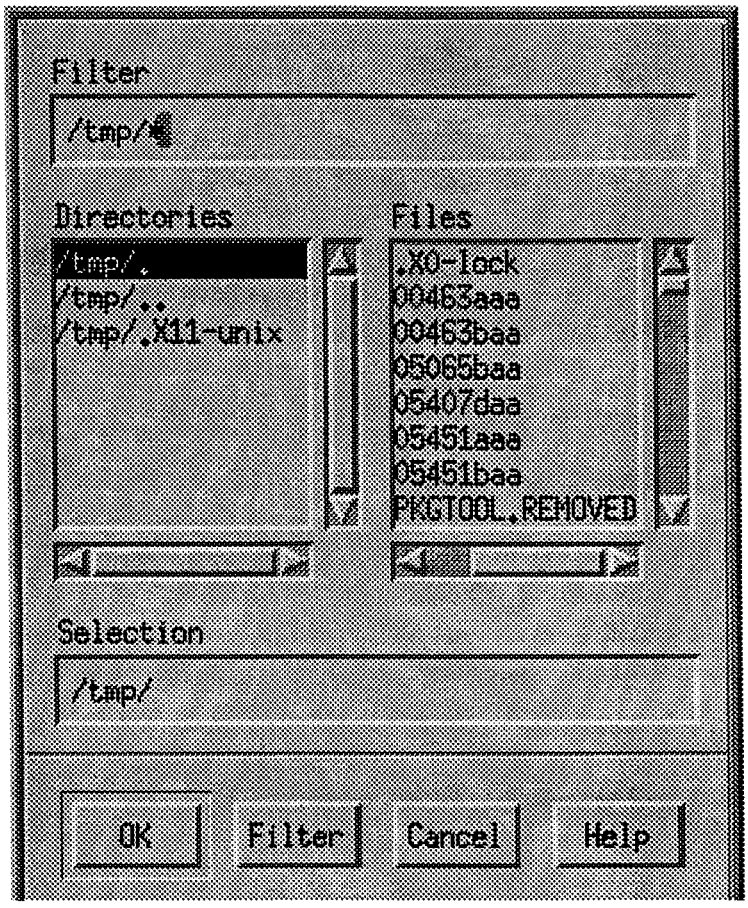

Figura 3.32: FileSelectionDialog

```
XtAddCallback(dialogo,XmNokCallback,maneja_ok,(XtPointer)2);
for (n=0;n<NUM_VALORES;n++)
  XmStringFree(valores[n]);
XmStringFree(xmcad);
XmStringFree(camino);
```

Como estos tres últimos widgets comentados llaman a la misma rutina cuando se pulsa el botón Ok, con distinto parámetro de usuario, a continuación se muestra el código de la misma.

```
void maneja_ok(Widget w, XtPointer client_data, XtPointer call_data)
{
  int opcion=(int)client_data;
  char *cadena;
  XmFileSelectionBoxCallbackStruct *ok_structura=
                    (XmFileSelectionBoxCallbackStruct *)call_data;
  XmStringGetLtoR(ok_structura->value,char_set,&cadena);
  switch (opcion) {
  case 1:
    printf("Fichero Selecionado: %s\n",cadena);
    XtDestroyWidget(w);
    break;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

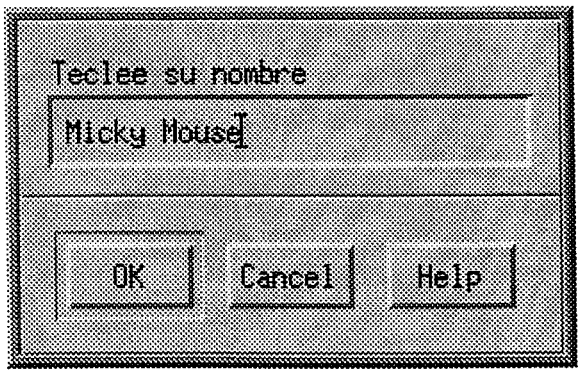

Figura 3.33: PromptDialog

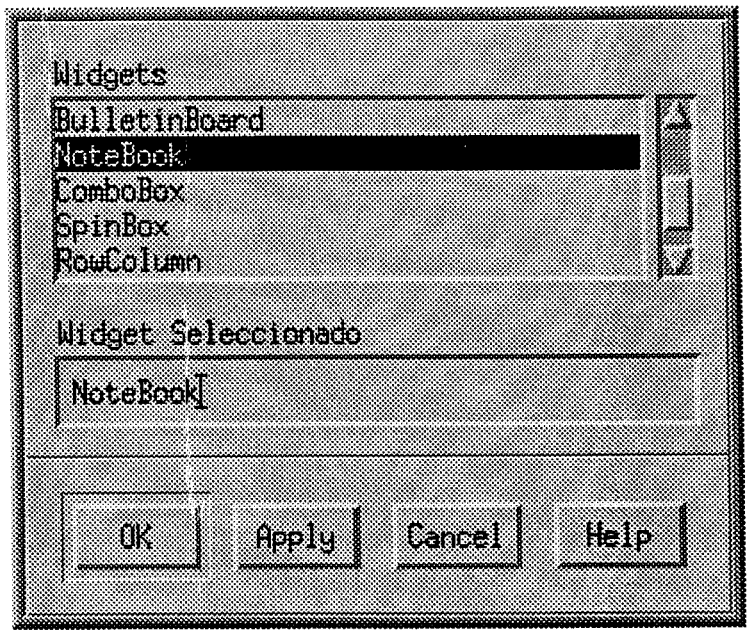

Figura 3.34: SelectionDialog

```
  case 2:
    printf("Widget Selecionado: %s\n",cadena);
    break;
  case 3:
    printf("Su nombre es: %s\n",cadena);
  }
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Apéndice

aloja_color.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#define COLOR 5

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 XWindowAttributes win_att;
 int x,y;

 XSetWindowAttributes atributos;

 XImage          *imag;
 XColor colores[COLOR];
 Colormap cmap;

 int cont=0,i;
 int plane_mask[1];
 int colors[COLOR];
 int ncolors=COLOR;
 char nombre[5][15];

 strcpy(nombre[0],"violet");
 strcpy(nombre[1],"salmon");
 strcpy(nombre[2],"coral");
 strcpy(nombre[3],"aquamarine");
 strcpy(nombre[4],"darkorchid");

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 cmap=DefaultColormap(dpy,scr);

 printf("Celdas de color %d, Profundidad %d\n",XDisplayCells(dpy,scr),
 XDisplayPlanes(dpy,scr));

 thewin.height=200;
 thewin.width=200;
```


© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 atributos.background_pixel=back;
 atributos.border_pixel=fore;
 atributos.colormap=cmap;

 win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
8,InputOutput,DefaultVisual(dpy,scr),
CWColormap | CWBackPixel | CWBorderPixel, &atributos);

 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);

 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);

/* Atributos de la ventana */
 XGetWindowAttributes(dpy,win,&win_att);

 XSelectInput(dpy,win,ButtonPressMask | KeyPressMask | ExposureMask);

/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);


 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
   case Expose:
                          imag=XGetImage(dpy,win,0,0,thewin.width,thewin.height,
AllPlanes,XYPixmap);
        break;
   case KeyPress:
XFreeGC(dpy,gc);
        XDestroyWindow(dpy,win);
        XCloseDisplay(dpy);
        exit(1);
break;
  case ButtonPress:
    switch(ev.xbutton.button){
     case Button1:
      XPutPixel(imag,ev.xbutton.x,ev.xbutton.y,fore);
      XPutImage(dpy,win,gc,imag,0,0,0,0,thewin.width,
thewin.height);
     break;
     case Button2:

      /* Hay que vigilar si no puede alojar ncolors */
      if (!XAllocColorCells(dpy,cmap,False,plane_mask,0,
colors,ncolors))
 printf("Error\n");

      for (i=0;i<5;i++){
        if (!XParseColor(dpy,cmap,nombre[i],colores[i]))
printf("Error Parse\n");
colores[i].flags=DoRed | DoGreen | DoBlue;
colores[i].pixel=colors[i];
      }
      XStoreColors(dpy,cmap,colores,ncolors);
     break;
     case Button3:
      cont++;
      if (cont==5) cont=0;
      fore=colores[cont].pixel;
     break;

     }
  break;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
}
}
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

apilado.c

```
#include <Xm/Xm.h>
#include <Xm/PanedW.h>
#include <Xm/PushB.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,apilado,boton1,boton2;

void main(int, char **);
void maneja_boton(Widget, XtPointer, XtPointer);

void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Boton pulsado\n");
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  char *cadena=NULL;

  toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  apilado=XmCreatePanedWindow(toplevel,"apilado",args,n);
  XtManageChild(apilado);

  n=0;
  XtSetArg(args[n],XmNwidth,100); n++;
  XtSetArg(args[n],XmNheight,75); n++;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-1", char_set));
n++;
  boton1=XmCreatePushButton(apilado,"boton",args,n);
  XtManageChild(boton1);

  n=0;
  XtSetArg(args[n],XmNwidth,100); n++;
  XtSetArg(args[n],XmNheight,75); n++;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-2", char_set));
n++;
  boton2=XmCreatePushButton(apilado,"boton",args,n);
  XtManageChild(boton2);

  XtAddCallback(boton1,XmNactivateCallback,maneja_boton,NULL);
  XtAddCallback(boton2,XmNactivateCallback,maneja_boton,NULL);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

backing_store.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

#define  TRUE 1
#define  FALSE 0

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 XSetWindowAttributes atributos;

 int backing;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;

 /* saving_unders & backing_store */
 if (DoesSaveUnders(DefaultScreenOfDisplay(dpy))){
    printf("Tengo save_under\n");
    atributos.save_under=TRUE;
    }
    else
     atributos.save_under=FALSE;

 atributos.backing_store=DoesBackingStore(DefaultScreenOfDisplay(dpy));
 if (atributos.backing_store==NotUseful) printf("NotUseful\n");
 if (atributos.backing_store==WhenMapped) printf("WhenMapped\n");
 if (atributos.backing_store==Always) printf("Always\n");

 atributos.background_pixmap=ParentRelative;
 atributos.border_pixel=fore;

 win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
DefaultDepth(dpy,scr),InputOutput,DefaultVisual(dpy,scr),
CWBackPixmap || CWBorderPixel || CWSaveUnder || CWBackingStore,
&atributos);

 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);

 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);

 XSelectInput(dpy,win,ButtonPressMask);

/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
    printf("Expose\n");
  break;
  case ButtonPress:
XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  break;
}
}
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

block.c

```
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/Notebook.h>
#include <Xm/Label.h>
#include <Xm/PushB.h>

#define PAG_BLOCK 11

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,frame,block,label,boton;

void main(int, char **);

void main(int argc, char *argv[])
{
  register int i;
  Arg args[16];
  register int n;
  char *cadena[PAG_BLOCK+1]={"vacio",
                             "COCHES",
                             "Seat 600",
                             "VW escarabajo",
                             "Renault",
                             "Porche",
                             "BMW",
                             "Ferrari",
                             "Ford",
                             "Cadillac",
                             "Jeep",
                             "Pontiac"
                            };
  toplevel=XtAppInitialize(&app_context,"block",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNwidth,300); n++;
  XtSetArg(args[n],XmNheight,350); n++;
  block=XmCreateNotebook(toplevel,"block",args,n);
  XtManageChild(block);

  for (i=1;i<=PAG_BLOCK;i++) {
    n=0;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
    XtSetArg(args[n],XmNnotebookChildType,XmPAGE); n++;
    XtSetArg(args[n],XmNpageNumber,i);
    frame=XmCreateFrame(block,"frame",args,n);
    XtManageChild(frame);

    n=0;
    XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR(cadena[i], char_set));
n++;
    label=XmCreateLabel(frame,"label",args,n);
    XtManageChild(label);
  }

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Europeos", char_set));
n++;
  XtSetArg(args[n],XmNnotebookChildType,XmSTATUS_AREA); n++;
  XtSetArg(args[n],XmNpageNumber,2); n++;
  label=XmCreateLabel(block,"label",args,n);
  XtManageChild(label);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Europeos", char_set));
n++;
  XtSetArg(args[n],XmNnotebookChildType,XmMAJOR_TAB); n++;
  XtSetArg(args[n],XmNpageNumber,1); n++;
  boton=XmCreatePushButton(block,"boton",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Utilitarios", char_set));
 n++;
  XtSetArg(args[n],XmNnotebookChildType,XmMINOR_TAB); n++;
  XtSetArg(args[n],XmNpageNumber,2); n++;
  boton=XmCreatePushButton(block,"boton",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Deportivos", char_set));
n++;
  XtSetArg(args[n],XmNnotebookChildType,XmMINOR_TAB); n++;
  XtSetArg(args[n],XmNpageNumber,5); n++;
  boton=XmCreatePushButton(block,"boton",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Americanos", char_set));
n++;
  XtSetArg(args[n],XmNnotebookChildType,XmMAJOR_TAB); n++;
  XtSetArg(args[n],XmNpageNumber,8); n++;
  boton=XmCreatePushButton(block,"boton",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Utilitarios", char_set));
 n++;
  XtSetArg(args[n],XmNnotebookChildType,XmMINOR_TAB); n++;
  XtSetArg(args[n],XmNpageNumber,8); n++;
  boton=XmCreatePushButton(block,"boton",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Deportivos", char_set));
n++;
  XtSetArg(args[n],XmNnotebookChildType,XmMINOR_TAB); n++;
  XtSetArg(args[n],XmNpageNumber,11); n++;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  boton=XmCreatePushButton(block,"boton",args,n);
  XtManageChild(boton);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

boton.c

```
#include <Xm/Xm.h>
#include <Xm/PushB.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,button;

void main(int, char **);
void maneja_boton(Widget, XtPointer, XtPointer);

void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Boton pulsado\n");
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  char *cadena=NULL;

  toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Pulsar Aqui", char_set));
 n++;
  button=XmCreatePushButton(toplevel,"boton",args,n);
  XtManageChild(button);

  XtAddCallback(button,XmNactivateCallback,maneja_boton,NULL);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

botones.c

```
#include <X11/X.h>
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>
#include <Xm/DrawnB.h>
#include <Xm/ArrowB.h>
#include <Xm/Separator.h>
#include <Xm/RowColumn.h>
#include <Xm/ToggleB.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,formulario,boton1,boton2,boton3,separador,radio,check;

void main(int, char **);
void maneja_boton1(Widget, XtPointer, XtPointer);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
void maneja_boton2(Widget, XtPointer, XtPointer);
void maneja_boton3(Widget, XtPointer, XtPointer);

void maneja_boton1(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Se ha pulsado el PushButton\n");
}

void maneja_boton2(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Se ha pulsado el ArrowButton\n");
}

void maneja_boton3(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Se ha pulsado el DrawnButton\n");
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  Pixmap camello;
  Display *display;
  char *fichero="camello.bmp";
  int ancho,alto,x_hot,y_hot;

  toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNwidth,260); n++;
  XtSetArg(args[n],XmNheight,200); n++;
  XtSetArg(args[n],XmNverticalSpacing,10); n++;
  XtSetArg(args[n],XmNhorizontalSpacing,10); n++;
  formulario=XmCreateForm(toplevel,"formulario",args,n);
  XtManageChild(formulario);

/* Creacion del PushButton */
  n=0;
  XtSetArg(args[n],XmNheight,58); n++;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Push Button", char_set));
 n++;
  boton1=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton1);

  XtAddCallback(boton1,XmNactivateCallback,maneja_boton1,NULL);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
  XtSetValues(boton1,args,n);

/* Creacion del ArrowButton */

  n=0;
  XtSetArg(args[n],XmNwidth,58); n++;
  XtSetArg(args[n],XmNheight,58); n++;
  XtSetArg(args[n],XmNarrowDirection,XmARROW_DOWN); n++;
  boton2=XmCreateArrowButton(formulario,"boton",args,n);
  XtManageChild(boton2);

  XtAddCallback(boton2,XmNactivateCallback,maneja_boton2,NULL);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,boton1); n++;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  XtSetValues(boton2,args,n);
/* Creacion del DrawnButton */
  display=XtDisplay(toplevel);
  XReadBitmapFile(display,DefaultRootWindow(display),fichero,&ancho,&alto,
&camello,&x_hot,&y_hot);
  printf("ancho: %d alto: %d\n",ancho,alto);
  printf("x_hot: %d y_hot: %d\n",x_hot,y_hot);
  n=0;
  XtSetArg(args[n],XmNwidth,58); n++;
  XtSetArg(args[n],XmNheight,58); n++;
  XtSetArg(args[n],XmNlabelType,XmPIXMAP); n++;
  XtSetArg(args[n],XmNpushButtonEnabled,True); n++;
  XtSetArg(args[n],XmNlabelPixmap,camello); n++;
  boton3=XmCreateDrawnButton(formulario,"boton",args,n);
  XtManageChild(boton3);
  XtAddCallback(boton3,XmNactivateCallback,maneja_boton3,NULL);
  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,boton2); n++;
  XtSetValues(boton3,args,n);
/* Separador debajo de los botones */
  n=0;
  separador=XmCreateSeparator(formulario,"separador",args,n);
  XtManageChild(separador);
  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNtopWidget,boton1); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
  XtSetValues(separador,args,n);
/* RadioBox */
  n=0;
  XtSetArg(args[n],XmNisHomogeneous,True); n++;
  XtSetArg(args[n],XmNpacking,XmPACK_COLUMN); n++;
  XtSetArg(args[n],XmNradioBehavior,True); n++;
  XtSetArg(args[n],XmNrowColumnType,XmWORK_AREA); n++;
  radio=XmCreateRowColumn(formulario,"radio",args,n);
  XtManageChild(radio);
  n=0;
  XtSetArg(args[n],XmNindicatorOn,XmINDICATOR_CROSS); n++;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Toggle Button 1",
char_set)); n++;
  boton1=XmCreateToggleButton(radio,"boton",args,n);
  XtManageChild(boton1);
  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Toggle Button 2",
char_set)); n++;
  boton1=XmCreateToggleButton(radio,"boton",args,n);
  XtManageChild(boton1);
  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Toggle Button 3",
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
char_set)); n++;
  boton1=XmCreateToggleButton(radio,"boton",args,n);
  XtManageChild(boton1);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNtopWidget,separador); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
  XtSetValues(radio,args,n);

/* CheckBox */

  n=0;
  check=XmCreateRowColumn(formulario,"radio",args,n);
  XtManageChild(check);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Toggle Button 1",
char_set)); n++;
  boton1=XmCreateToggleButton(check,"boton",args,n);
  XtManageChild(boton1);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Toggle Button 2",
char_set)); n++;
  boton1=XmCreateToggleButton(check,"boton",args,n);
  XtManageChild(boton1);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Toggle Button 3",
char_set)); n++;
  boton1=XmCreateToggleButton(check,"boton",args,n);
  XtManageChild(boton1);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNtopWidget,separador); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
  XtSetValues(check,args,n);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

colormap.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xatom.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 XWindowAttributes win_att;
 int x,y;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
XSetWindowAttributes atributos;
Colormap cmap;
XStandardColormap info;

XImage          *imag;
unsigned long col=1;


dpy=XOpenDisplay("");
scr=DefaultScreen(dpy);
fore=WhitePixel(dpy,scr);
back=BlackPixel(dpy,scr);

XGetStandardColormap(dpy,DefaultRootWindow(dpy),&info,XA_RGB_DEFAULT_MAP);

cmap=info.colormap;

printf("Celdas de color %d, Profundidad %d\n",XDisplayCells(dpy,scr),
XDisplayPlanes(dpy,scr));

thewin.height=200;
thewin.width=200;
atributos.background_pixel=back;
atributos.border_pixel=fore;
atributos.colormap=cmap;

win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
8,InputOutput,DefaultVisual(dpy,scr),
CWColormap | CWBackPixel | CWBorderPixel, &atributos);

XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);

gc = XCreateGC(dpy, win, 0, NULL);
XSetBackground(dpy,gc,back);
XSetForeground(dpy,gc,fore);

/* Atributos de la ventana */
XGetWindowAttributes(dpy,win,&win_att);

XSelectInput(dpy,win,ButtonPressMask | KeyPressMask | ExposureMask);

/* Map the window, wait for the Expose events, and paint */
XMapWindow(dpy, win);


while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
   case Expose:
                        imag=XGetImage(dpy,win,0,0,thewin.width,thewin.height,
AllPlanes,XYPixmap);
        break;
   case KeyPress:
XFreeGC(dpy,gc);
        XDestroyWindow(dpy,win);
        XCloseDisplay(dpy);
        exit(1);
break;
  case ButtonPress:
      XPutPixel(imag,ev.xbutton.x,ev.xbutton.y,fore);
      XPutImage(dpy,win,gc,imag,0,0,0,0,thewin.width,
thewin.height);
  break;
}
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

combo.c

```
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/BulletinB.h>
#include <Xm/ComboBox.h>

#define NUM_ITEMS   6
XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,marco,tablon,combo;

void main(int, char **);
void maneja_boton(Widget, XtPointer, XtPointer);

void maneja_seleccion(Widget w, XtPointer client_data, XtPointer call_data)
{
  XmComboBoxCallbackStruct *combo_struct =
                                        (XmComboBoxCallbackStruct *) call_data;
  char *cadena;

  XmStringGetLtoR(combo_struct->item_or_text,char_set,&cadena);

  printf("Seleccion: %s\n",cadena);
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  XmString xmcad[NUM_ITEMS];
  char *cadena[]={"Seat", "Porsche", "BMW", "Mercedes", "Fiat", "Ferrari"};

  toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  marco=XmCreateFrame(toplevel,"",args,n);
  XtManageChild(marco);

  n=0;
  XtSetArg(args[n],XmNheight,200); n++;
  XtSetArg(args[n],XmNwidth,200); n++;
  tablon=XmCreateBulletinBoard(marco,"",args,n);
  XtManageChild(tablon);

  for (n=0;n<NUM_ITEMS;n++)
    xmcad[n]=XmStringCreateLtoR(cadena[n],char_set);

  n=0;
  XtSetArg(args[n],XmNwidth,150); n++;
  XtSetArg(args[n],XmNcomboBoxType,XmDROP_DOWN_COMBO_BOX); n++;
  XtSetArg(args[n],XmNarrowSpacing,5); n++;
  XtSetArg(args[n],XmNitems,xmcad); n++;
  XtSetArg(args[n],XmNitemCount,NUM_ITEMS); n++;
  XtSetArg(args[n],XmNvisibleItemCount,4); n++;
  combo=XmCreateComboBox(tablon,"combo",args,n);
  XtManageChild(combo);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  XtAddCallback(combo,XmNselectionCallback,maneja_seleccion,NULL);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

contenedor.c

```
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/Container.h>
#include <Xm/IconG.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,frame,contenedor,entrada1,entrada2,entrada3,entrada11,
entrada12, entrada21,entrada22;

void main(int, char **);
void maneja_boton(Widget, XtPointer, XtPointer);

void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Boton pulsado\n");
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  char *cadena=NULL;
  XmString xmstr[5];

  toplevel=XtAppInitialize(&app_context,"contenedor",NULL,0,&argc,argv,
NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNmarginWidth,10); n++;
  XtSetArg(args[n],XmNmarginHeight,10); n++;
  frame=XmCreateFrame(toplevel,"frame",args,n);
  XtManageChild(frame);

  n=0;
  XtSetArg(args[n],XmNlayoutType,XmDETAIL); n++;
  XtSetArg(args[n],XmNselectionPolicy,XmBROWSE_SELECT); n++;
  XtSetArg(args[n],XmNautomaticSelection,XmNO_AUTO_SELECT); n++;
  XtSetArg(args[n],XmNentryViewType,XmLARGE_ICON); n++;
  contenedor=XmCreateContainer(frame,"contenedor",args,n);
  XtManageChild(contenedor);

  XtAddCallback(contenedor,XmNdefaultActionCallback,maneja_boton,
(XtPointer)NULL);

  xmstr[0]=XmStringCreateLtoR("Edif. Principal", char_set);
  xmstr[1]=XmStringCreateLtoR("jph@aaa.es", char_set);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Del. Canarias",
char_set)); n++;
  XtSetArg(args[n],XmNdetail,xmstr); n++;
  XtSetArg(args[n],XmNdetailCount,2); n++;
  XtSetArg(args[n],XmNoutlineState,XmEXPANDED); n++;
  entrada1=XmCreateIconGadget(contenedor,"entrada",args,n);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  XtManageChild(entrada1);

  xmstr[0]=XmStringCreateLtoR("C. Triana", char_set);
  xmstr[1]=XmStringCreateLtoR("ohr@aaa.es", char_set);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Subdel. Las Palmas",
char_set)); n++;
  XtSetArg(args[n],XmNentryParent,entrada1); n++;
  XtSetArg(args[n],XmNdetail,xmstr); n++;
  XtSetArg(args[n],XmNdetailCount,2); n++;
  entrada11=XmCreateIconGadget(contenedor,"entrada",args,n);
  XtManageChild(entrada11);

  xmstr[0]=XmStringCreateLtoR("C. Castillo", char_set);
  xmstr[1]=XmStringCreateLtoR("rgp@aab.es", char_set);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Subdel. Tenerife",
char_set)); n++;
  XtSetArg(args[n],XmNdetail,xmstr); n++;
  XtSetArg(args[n],XmNdetailCount,2); n++;
  XtSetArg(args[n],XmNentryParent,entrada1); n++;
  entrada12=XmCreateIconGadget(contenedor,"entrada",args,n);
  XtManageChild(entrada12);

  xmstr[0]=XmStringCreateLtoR("Edif. Bahia", char_set);
  xmstr[1]=XmStringCreateLtoR("ezs@bbb.es", char_set);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Del. Extremadura",
char_set)); n++;
  XtSetArg(args[n],XmNdetail,xmstr); n++;
  XtSetArg(args[n],XmNdetailCount,2); n++;
  entrada2=XmCreateIconGadget(contenedor,"entrada",args,n);
  XtManageChild(entrada2);

  xmstr[0]=XmStringCreateLtoR("C. Verde", char_set);
  xmstr[1]=XmStringCreateLtoR("mhw@bbb.es", char_set);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Subdel. Caceres",
char_set)); n++;
  XtSetArg(args[n],XmNentryParent,entrada2); n++;
  XtSetArg(args[n],XmNdetail,xmstr); n++;
  XtSetArg(args[n],XmNdetailCount,2); n++;
  entrada21=XmCreateIconGadget(contenedor,"entrada",args,n);
  XtManageChild(entrada21);

  xmstr[0]=XmStringCreateLtoR("C. Azul", char_set);
  xmstr[1]=XmStringCreateLtoR("qjl@bbb.es", char_set);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Subdel. Badajoz",
char_set)); n++;
  XtSetArg(args[n],XmNentryParent,entrada2); n++;
  XtSetArg(args[n],XmNdetail,xmstr); n++;
  XtSetArg(args[n],XmNdetailCount,2); n++;
  entrada22=XmCreateIconGadget(contenedor,"entrada",args,n);
  XtManageChild(entrada22);

  xmstr[0]=XmStringCreateLtoR("C. Gran Via", char_set);
  xmstr[1]=XmStringCreateLtoR("lvd@ccc.es", char_set);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Del. Madrid", char_set));
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 n++;
  XtSetArg(args[n],XmNdetail,xmstr); n++;
  XtSetArg(args[n],XmNdetailCount,2); n++;
  entrada3=XmCreateIconGadget(contenedor,"entrada",args,n);
  XtManageChild(entrada3);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

cursor.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/cursorfont.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win,win2;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin,thewin2;
 XEvent ev;
 Cursor cursor;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 thewin2.height=100;
 thewin2.width=100;
 win2 = XCreateSimpleWindow(dpy, win,
0, 0, thewin2.width, thewin2.height, 5, fore,back);
 XSetStandardProperties(dpy,win2,"Ventana saludo II","Hola II",None,
argv,argc,&thewin2);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 /*Cursor*/
 cursor=XCreateFontCursor(dpy,XC_dot);
 XDefineCursor(dpy,win2,cursor);
 XSelectInput(dpy,win,ExposureMask | KeyPressMask);
 XSelectInput(dpy,win2,ExposureMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 XMapWindow(dpy, win2);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0){
        if (ev.xexpose.window==win)
  XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  else
  XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
30,45,"Hola Mundillo",13);
}
  break;
  case KeyPress:
       XFreeGC(dpy,gc);
       XDestroyWindow(dpy,win);
       XCloseDisplay(dpy);
     exit(1);
  break;
  default:
  break;
}
}
}
```

cursor_propio.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/cursorfont.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win,win2;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin,thewin2;
 XEvent ev;
 Cursor cursor;
 Pixmap ima;
 XColor Fore,Back;
 Colormap cmap;

#define XC_circle_width 16
#define XC_circle_height 16
static char XC_circle_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0xe0, 0x03, 0xf8, 0x0f, 0xfc, 0x1f, 0xfc, 0x1f,
   0xfe, 0x3f, 0xfe, 0x3f, 0xfe, 0x3f, 0xfe, 0x3f, 0xfe, 0x3f, 0xfc, 0x1f,
   0xfc, 0x1f, 0xf8, 0x0f, 0xe0, 0x03, 0x00, 0x00};

 dpy=XOpenDisplay("");
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 scr=DefaultScreen(dpy);
 cmap=DefaultColormap(dpy,scr);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 thewin2.height=100;
 thewin2.width=100;
 win2 = XCreateSimpleWindow(dpy, win,
0, 0, thewin2.width, thewin2.height, 5, fore,back);
 XSetStandardProperties(dpy,win2,"Ventana saludo II","Hola II",None,
argv,argc,&thewin2);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 /*Cursor*/
 XAllocNamedColor(dpy,cmap,"white",&Fore,&Fore);
 XAllocNamedColor(dpy,cmap,"black",&Back,&Back);
 ima=XCreatePixmapFromBitmapData(dpy,DefaultRootWindow(dpy), XC_circle_bits,
 XC_circle_width,XC_circle_height, fore,back,DefaultDepth(dpy,scr));
 cursor=XCreatePixmapCursor(dpy,ima,None,Fore,Back,0,0);
 XDefineCursor(dpy,win2,cursor);
 XSelectInput(dpy,win,ExposureMask | KeyPressMask);
 XSelectInput(dpy,win2,ExposureMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);
 XMapWindow(dpy, win2);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0){
        if (ev.xexpose.window==win)
  XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  else
  XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
30,45,"Hola Mundillo",13);
}
  break;
  case KeyPress:
       XFreeGC(dpy,gc);
       XDestroyWindow(dpy,win);
       XCloseDisplay(dpy);
     exit(1);
  break;
  default:
  break;
}
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
}
```

dialogos.c

```
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>
#include <Xm/MessageB.h>
#include <Xm/FileSB.h>
#include <Xm/SelectioB.h>

#define NUM_VALORES  10

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,marco,filacol,boton[8];
char *cad_valores[]={"PushButton", "CascadeButton", "ToggleButton", "Form",
                     "BulletinBoard", "NoteBook", "ComboBox", "SpinBox",
     "RowColumn", "List" };

void main(int, char **);
void maneja_boton(Widget, XtPointer, XtPointer);
void maneja_ok(Widget, XtPointer, XtPointer);

void main(int argc, char *argv[])
{
  register int i;
  Arg args[16];
  register int n;
  char *cadena[8]={"Error", "Informacion", "Aviso", "Trabajando", "Ficheros",
     "Seleccion", "Pregunta", "Sugerencia"};

  toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNwidth,300); n++;
  marco=XmCreateFrame(toplevel,"",args,n);
  XtManageChild(marco);

  n=0;
  XtSetArg(args[n],XmNnumColumns,3); n++;
  XtSetArg(args[n],XmNpacking,XmPACK_COLUMN); n++;
  filacol=XmCreateRowColumn(marco,"filacol",args,n);
  XtManageChild(filacol);

  for (i=0;i<8;i++) {
    n=0;
    XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR(cadena[i], char_set));
n++;
    boton[i]=XmCreatePushButton(filacol,"boton",args,n);
    XtManageChild(boton[i]);

    XtAddCallback(boton[i],XmNactivateCallback,maneja_boton,(XtPointer)(i+1));
  }

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  int opcion=(int)client_data;
  Arg args[16];
  register int n;
  Widget dialogo;
  XmString xmcad,camino,mascara,valores[NUM_VALORES];

  printf("opcion: %d\n",opcion);
  switch (opcion) {
  case 1:
    xmcad=XmStringCreateLtoR("Esto es un error que se ha producido",char_set);

    n=0;
    XtSetArg(args[n],XmNmessageString,xmcad); n++;
    dialogo=XmCreateErrorDialog(toplevel,"",args,n);
    XtManageChild(dialogo);

      XmStringFree(xmcad);
      break;
    case 2:
      xmcad=XmStringCreateLtoR("Esto es una informacion para el usuario",
char_set);

      n=0;
      XtSetArg(args[n],XmNmessageString,xmcad); n++;
      dialogo=XmCreateInformationDialog(toplevel,"",args,n);
      XtManageChild(dialogo);

      XmStringFree(xmcad);
      break;
    case 3:
      xmcad=XmStringCreateLtoR("Atencion la opcion no es valida",char_set);
      n=0;
      XtSetArg(args[n],XmNmessageString,xmcad); n++;
      dialogo=XmCreateWarningDialog(toplevel,"",args,n);
      XtManageChild(dialogo);

      XmStringFree(xmcad);
      break;
    case 4:
      xmcad=XmStringCreateLtoR("Sin apuros, que estoy trabajando",char_set);

      n=0;
      XtSetArg(args[n],XmNmessageString,xmcad); n++;
      dialogo=XmCreateWorkingDialog(toplevel,"",args,n);
      XtManageChild(dialogo);

      XmStringFree(xmcad);
      break;
    case 5:
      camino=XmStringCreateLtoR("/tmp",char_set);
      mascara=XmStringCreateLtoR("*",char_set);

      n=0;
      XtSetArg(args[n],XmNdirectory,camino); n++;
      XtSetArg(args[n],XmNpattern,mascara); n++;
      dialogo=XmCreateFileSelectionDialog(toplevel,"",args,n);
      XtManageChild(dialogo);

      XtAddCallback(dialogo,XmNokCallback,maneja_ok,(XtPointer)1);

      XmStringFree(camino);
      XmStringFree(mascara);
      break;
    case 6:
      for (n=0;n<NUM_VALORES;n++)
  valores[n]=XmStringCreateLtoR(cad_valores[n],char_set);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
      xmcad=XmStringCreateLtoR("Widgets",char_set);
      camino=XmStringCreateLtoR("Widget Seleccionado",char_set);

      n=0;
      XtSetArg(args[n],XmNlistItems,valores); n++;
      XtSetArg(args[n],XmNlistItemCount,NUM_VALORES); n++;
      XtSetArg(args[n],XmNlistVisibleItemCount,5); n++;
      XtSetArg(args[n],XmNlistLabelString,xmcad); n++;
      XtSetArg(args[n],XmNselectionLabelString,camino); n++;
      dialogo=XmCreateSelectionDialog(toplevel,"",args,n);
      XtManageChild(dialogo);

      XtAddCallback(dialogo,XmNokCallback,maneja_ok,(XtPointer)2);

      for (n=0;n<NUM_VALORES;n++)
XmStringFree(valores[n]);
      XmStringFree(xmcad);
      XmStringFree(camino);
      break;
    case 7:
      xmcad=XmStringCreateLtoR("Cree que los ordenadores solucionan algo?",
char_set);
      n=0;
      XtSetArg(args[n],XmNmessageString,xmcad); n++;
      dialogo=XmCreateQuestionDialog(toplevel,"",args,n);
      XtManageChild(dialogo);

      XmStringFree(xmcad);
      break;
    case 8:
      xmcad=XmStringCreateLtoR("Teclee su nombre",char_set);
      n=0;
      XtSetArg(args[n],XmNselectionLabelString,xmcad); n++;
      dialogo=XmCreatePromptDialog(toplevel,"",args,n);
      XtManageChild(dialogo);

      XtAddCallback(dialogo,XmNokCallback,maneja_ok,(XtPointer)3);

      XmStringFree(xmcad);
      break;
  }
}

void maneja_ok(Widget w, XtPointer client_data, XtPointer call_data)
{
  int opcion=(int)client_data;
  char *cadena;
  XmFileSelectionBoxCallbackStruct *ok_structura=(XmFileSelectionBoxCallbackStruct *)ca

  XmStringGetLtoR(ok_structura->value,char_set,&cadena);

  switch (opcion) {
  case 1:
    printf("Fichero Selecionado: %s\n",cadena);
    XtDestroyWidget(w);
    break;
  case 2:
    printf("Widget Selecionado: %s\n",cadena);
    break;
  case 3:
    printf("Su nombre es: %s\n",cadena);
  }
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

escala.c

```
#include <stdio.h>
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/Scale.h>
#include <Xm/Label.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,marco,escala,etiqueta[11];

void main(int, char **);
void maneja_escala(Widget, XtPointer, XtPointer);

void maneja_escala(Widget w, XtPointer client_data, XtPointer call_data)
{
  int valor,razon;
  XmScaleCallbackStruct *call_value=(XmScaleCallbackStruct *)call_data;

  razon=call_value->reason;
  valor=call_value->value;

  switch (razon) {
  case XmCR_VALUE_CHANGED:
    printf("Razon: XmCR_VALUE_CHANGED, valor: %d\n",valor);
    break;
  case XmCR_DRAG:
    printf("Razon: XmCR_DRAG, valor: %d\n",valor);
    break;
  }
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  char valor_cad[10],*cadena="Demostracion del Widget Scale";
  XmString xmcad;
  register int i;

  toplevel=XtAppInitialize(&app_context,"escala",NULL,0,&argc,argv,NULL,
NULL,0);

  xmcad=XmStringCreateLtoR(cadena,char_set);

  n=0;
  XtSetArg(args[n],XmNmarginWidth,20); n++;
  XtSetArg(args[n],XmNmarginHeight,20); n++;
  marco=XmCreateFrame(toplevel,"frame",args,n);
  XtManageChild(marco);

  n=0;
  XtSetArg(args[n],XmNshowValue,True); n++;
  XtSetArg(args[n],XmNtitleString,xmcad); n++;
  XtSetArg(args[n],XmNorientation,XmHORIZONTAL); n++;
  XtSetArg(args[n],XmNmaximum,100); n++;
  escala=XmCreateScale(marco,"escala",args,n);
  XtManageChild(escala);

  XtAddCallback(escala,XmNvalueChangedCallback,maneja_escala,NULL);
  XtAddCallback(escala,XmNdragCallback,maneja_escala,NULL);

  for (i=0;i<=10;i++) {
    n=0;
    sprintf(valor_cad,"%d",i*10);
    etiqueta[i]=XmCreateLabel(escala,valor_cad,args,n);
  }
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  XtManageChildren(etiqueta,i);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

etiqueta.c

```
#include <Xm/Xm.h>
#include <Xm/BulletinB.h>
#include <Xm/Label.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,tablon,etiqueta1,etiqueta2,etiqueta3;

void main(int, char **);
void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  char *cadena=NULL;

  toplevel=XtAppInitialize(&app_context,"etiqueta",NULL,0,&argc,argv,
NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNwidth,110); n++;
  XtSetArg(args[n],XmNheight,120); n++;
  tablon=XmCreateBulletinBoard(toplevel,"tablon",args,n);
  XtManageChild(tablon);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Etiqueta-1",
char_set)); n++;
  XtSetArg(args[n],XmNx,1); n++;
  XtSetArg(args[n],XmNy,20); n++;
  XtSetArg(args[n],XmNwidth,90); n++;
  XtSetArg(args[n],XmNalignment,XmALIGNMENT_BEGINNING); n++;
  etiqueta1=XmCreateLabel(tablon,"etiqueta",args,n);
  XtManageChild(etiqueta1);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Etiqueta-2",
char_set)); n++;
  XtSetArg(args[n],XmNx,1); n++;
  XtSetArg(args[n],XmNy,60); n++;
  XtSetArg(args[n],XmNwidth,90); n++;
  XtSetArg(args[n],XmNalignment,XmALIGNMENT_CENTER); n++;
  etiqueta2=XmCreateLabel(tablon,"etiqueta",args,n);
  XtManageChild(etiqueta2);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Etiqueta-3",
char_set)); n++;
  XtSetArg(args[n],XmNx,1); n++;
  XtSetArg(args[n],XmNy,100); n++;
  XtSetArg(args[n],XmNwidth,90); n++;
  XtSetArg(args[n],XmNalignment,XmALIGNMENT_END); n++;
  etiqueta3=XmCreateLabel(tablon,"etiqueta",args,n);
  XtManageChild(etiqueta3);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

fila-columna.c

```
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/RowColumn.h>
#include <Xm/PushB.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,marco,filacol,boton[6];

void main(int, char **);

void main(int argc, char *argv[])
{
  register int i;
  Arg args[16];
  register int n;
  char *cadena[6]={"boton 1", "boton 2", "boton 3", "boton 4", "boton 5",
"boton 6"};

  toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNwidth,300); n++;
  marco=XmCreateFrame(toplevel,"",args,n);
  XtManageChild(marco);

  n=0;
  XtSetArg(args[n],XmNnumColumns,3); n++;
  XtSetArg(args[n],XmNpacking,XmPACK_COLUMN); n++;
  filacol=XmCreateRowColumn(marco,"filacol",args,n);
  XtManageChild(filacol);

  for (i=0;i<6;i++) {
    n=0;
    XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR(cadena[i],
char_set)); n++;
    boton[i]=XmCreatePushButton(filacol,"boton",args,n);
  }
  XtManageChildren(boton,6);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

formulario.c

```
#include <Xm/Xm.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,formulario,boton1,boton2,boton3,boton4;

void main(int, char **);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
void maneja_boton(Widget, XtPointer, XtPointer);

void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Boton pulsado\n");
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;

  toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,
NULL,0);

  n=0;
  XtSetArg(args[n],XmNwidth,200); n++;
  XtSetArg(args[n],XmNheight,100); n++;
  XtSetArg(args[n],XmNverticalSpacing,10); n++;
  XtSetArg(args[n],XmNhorizontalSpacing,10); n++;
  formulario=XmCreateForm(toplevel,"formulario",args,n);
  XtManageChild(formulario);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-1", char_set));
n++;
  boton1=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton1);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-2", char_set));
n++;
  boton2=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton2);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-3", char_set));
n++;
  boton3=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton3);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-4", char_set));
n++;
  boton4=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton4);

  XtAddCallback(boton1,XmNactivateCallback,maneja_boton,NULL);
  XtAddCallback(boton2,XmNactivateCallback,maneja_boton,NULL);
  XtAddCallback(boton3,XmNactivateCallback,maneja_boton,NULL);
  XtAddCallback(boton4,XmNactivateCallback,maneja_boton,NULL);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
  XtSetValues(boton1,args,n);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,boton1); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET); n++;
  XtSetValues(boton2,args,n);

  n=0;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  XtSetValues(boton3,args,n);
  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNtopWidget,boton2); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetValues(boton4,args,n);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

fuente.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 char **fuentes;
 int i,cont;
 Font ft;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
/* Recuperamos las fuentes */
 fuentes=XListFonts(dpy,"*bold*",100,&cont);
 for (i=0;i<cont;i++)
    printf("Tipo %s\n",fuentes[i]);
 ft=XLoadFont(dpy,fuentes[cont/2]);
 XFreeFontNames(fuentes);
 XSetFont(dpy,gc,ft);
 XSelectInput(dpy,win,ExposureMask | KeyPressMask);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0)
XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  break;
  case KeyPress:
       XFreeGC(dpy,gc);
       XUnloadFont(dpy,ft);
       XDestroyWindow(dpy,win);
       XCloseDisplay(dpy);
     exit(1);
  break;
  default:
  break;
}
}
}
```

hola-motif.c

```
#include <Xm/Xm.h>
#include <Xm/Label.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,label;

void  main(int, char **);

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;

  toplevel=XtAppInitialize(&app_context,"hola",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Hola Mundo", char_set));
n++;
  label=XmCreateLabel(toplevel,"etiqueta",args,n);

  XtManageChild(label);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

hola-xlib.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 XSelectInput(dpy,win,ExposureMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0)
XDrawString(ev.xexpose.display,ev.xexpose.window,
gc,70,95,"Hola Mundo",10);
  break;
}
}
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

marco.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
Display *dpy;
Window win;
int scr;
GC gc;
unsigned long fore,back;
XSizeHints thewin;
XEvent ev;
XWindowAttributes win_att;
int x,y;
XImage          *imag;

dpy=XOpenDisplay("");
scr=DefaultScreen(dpy);
fore=WhitePixel(dpy,scr);
back=BlackPixel(dpy,scr);

thewin.height=200;
thewin.width=200;
win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
gc = XCreateGC(dpy, win, 0, NULL);
XSetBackground(dpy,gc,back);
XSetForeground(dpy,gc,fore);
/* Atributos de la ventana */
XGetWindowAttributes(dpy,win,&win_att);
XSelectInput(dpy,win,ButtonPressMask | KeyPressMask);
/* Map the window, wait for the Expose events, and paint */
XMapWindow(dpy, win);

while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
   case KeyPress:
XFreeGC(dpy,gc);
        XDestroyWindow(dpy,win);
        XCloseDisplay(dpy);
        exit(1);
break;
  case ButtonPress:
    switch(ev.xbutton.button){
     case Button1:
      XPutPixel(imag,ev.xbutton.x,ev.xbutton.y,fore);
      XPutImage(dpy,win,gc,imag,0,0,0,0,thewin.width,
thewin.height);
     break;
     case Button2:
        imag=XGetImage(dpy,win,0,0,thewin.width,thewin.height,
AllPlanes,XYPixmap);
     break;
     }
  break;
}
}
XFreeGC(dpy,gc);
XDestroyWindow(dpy,win);
XCloseDisplay(dpy);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

}

marco.c

```
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/Form.h>
#include <Xm/PushB.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,marco,etiqueta,formulario,boton1,boton2,boton3,boton4;

void main(int, char **);
void maneja_boton(Widget, XtPointer, XtPointer);

void maneja_boton(Widget w, XtPointer client_data, XtPointer call_data)
{
  printf("Boton pulsado\n");
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;

  toplevel=XtAppInitialize(&app_context,"marco",NULL,0,&argc,argv,NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNmarginWidth,10); n++;
  XtSetArg(args[n],XmNmarginHeight,10); n++;
  marco=XmCreateFrame(toplevel,"frame",args,n);
  XtManageChild(marco);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Formulario", char_set));
n++;
  XtSetArg(args[n],XmNframeChildType,XmFRAME_TITLE_CHILD); n++;
  etiqueta=XmCreateLabel(marco,"etiqueta",args,n);
  XtManageChild(etiqueta);

  n=0;
  XtSetArg(args[n],XmNwidth,200); n++;
  XtSetArg(args[n],XmNheight,100); n++;
  XtSetArg(args[n],XmNverticalSpacing,10); n++;
  XtSetArg(args[n],XmNhorizontalSpacing,10); n++;
  XtSetArg(args[n],XmNframeChildType,XmFRAME_WORKAREA_CHILD); n++;
  formulario=XmCreateForm(marco,"formulario",args,n);
  XtManageChild(formulario);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-1", char_set));
n++;
  boton1=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton1);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-2", char_set));
n++;
  boton2=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton2);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-3", char_set));
n++;
  boton3=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton3);

  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Boton-4", char_set));
n++;
  boton4=XmCreatePushButton(formulario,"boton",args,n);
  XtManageChild(boton4);

  XtAddCallback(boton1,XmNactivateCallback,maneja_boton,NULL);
  XtAddCallback(boton2,XmNactivateCallback,maneja_boton,NULL);
  XtAddCallback(boton3,XmNactivateCallback,maneja_boton,NULL);
  XtAddCallback(boton4,XmNactivateCallback,maneja_boton,NULL);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
  XtSetValues(boton1,args,n);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNleftWidget,boton1); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_WIDGET); n++;
  XtSetValues(boton2,args,n);

  n=0;
  XtSetArg(args[n],XmNleftAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetValues(boton3,args,n);

  n=0;
  XtSetArg(args[n],XmNtopAttachment,XmATTACH_WIDGET); n++;
  XtSetArg(args[n],XmNtopWidget,boton2); n++;
  XtSetArg(args[n],XmNrightAttachment,XmATTACH_FORM); n++;
  XtSetArg(args[n],XmNbottomAttachment,XmATTACH_FORM); n++;
  XtSetValues(boton4,args,n);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

menus.c

```
#include <Xm/Xm.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/ScrolledW.h>
#include <Xm/PushB.h>
#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/Separator.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,vent_pral,tablon;

void main(int, char **);
void salir(Widget, XtPointer, XtPointer);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
Widget crear_menu_bar(Widget);
Widget crear_menu_popup(Widget);
Widget crear_menu_option(Widget);
void sacar_popup(Widget,XtPointer,XEvent *);

void salir(Widget w, XtPointer client_data, XtPointer call_data)
{
  XtUnmapWidget(toplevel);
  XtDestroyApplicationContext(XtWidgetToApplicationContext(toplevel));
  exit(0);
}

/* Creacion del menu */

Widget crear_menu_bar(Widget padre)
{
  Widget menub,menupane,boton,cascada;
  Arg args[16];
  int n;

  n=0;
  menub=XmCreateMenuBar(padre,"menu",args,n);
  XtManageChild(menub);

  n=0;
  menupane=XmCreatePulldownMenu(menub,"menu",args,n);

  n=0;
  boton=XmCreatePushButton(menupane,"Abrir",args,n);
  XtManageChild(boton);

  n=0;
  boton=XmCreatePushButton(menupane,"Guardar",args,n);
  XtManageChild(boton);

  n=0;
  boton=XmCreatePushButton(menupane,"Salir",args,n);
  XtManageChild(boton);

  XtAddCallback(boton,XmNactivateCallback,salir,NULL);

  n=0;
  XtSetArg(args[n],XmNsubMenuId,menupane); n++;
  cascada=XmCreateCascadeButton(menub,"Fichero",args,n);
  XtManageChild(cascada);

  n=0;
  menupane=XmCreatePulldownMenu(menub,"menu",args,n);

  n=0;
  boton=XmCreatePushButton(menupane,"Buscar",args,n);
  XtManageChild(boton);

  n=0;
  boton=XmCreatePushButton(menupane,"Sustituir",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNsubMenuId,menupane); n++;
  cascada=XmCreateCascadeButton(menub,"Editar",args,n);
  XtManageChild(cascada);

  n=0;
  menupane=XmCreatePulldownMenu(menub,"menu",args,n);

  n=0;
  menupane=XmCreatePulldownMenu(menub,"menu",args,n);

  n=0;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  boton=XmCreatePushButton(menupane,"Ayuda",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNsubMenuId,menupane); n++;
  cascada=XmCreateCascadeButton(menub,"Ayuda",args,n);
  XtManageChild(cascada);

  n=0;
  XtSetArg(args[n],XmNmenuHelpWidget,cascada); n++;
  XtSetValues(menub,args,n);

  return menub;
}

/* Rutina para crear el menu Popup */

Widget crear_menu_popup(Widget padre)
{
  Widget menupane,etiqueta,separador,boton,cascada;
  Arg args[16];
  int n;

  n=0;
  menupane=XmCreatePopupMenu(padre,"menu",args,n);

  n=0;
  etiqueta=XmCreateLabel(menupane,"Opciones",args,n);
  XtManageChild(etiqueta);

  n=0;
  separador=XmCreateSeparator(menupane,"",args,n);
  XtManageChild(separador);

  n=0;
  boton=XmCreatePushButton(menupane,"Normal",args,n);
  XtManageChild(boton);

  n=0;
  boton=XmCreatePushButton(menupane,"Negrilla",args,n);
  XtManageChild(boton);

  n=0;
  boton=XmCreatePushButton(menupane,"Cursiva",args,n);
  XtManageChild(boton);

  return menupane;
}

/* Rutina para mostrar el menu Popup */

void sacar_popup(Widget w, XtPointer client_data, XEvent *event)
{
  Widget menu_popup=(Widget)client_data;
  XButtonEvent *evento=(XButtonEvent *)event;

  if (evento->button == Button3) {
    XmMenuPosition(menu_popup,evento);
    XtManageChild(menu_popup);
  }
}

/* Rutina para crear el menu Option */

Widget crear_menu_option(Widget padre)
{
  Widget pulldown,menupane,boton,etiqueta,separador;
  XmString xmcad;
  Arg args[16];
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  int n;
  n=0;
  XtSetArg(args[n],XmNtearOffModel,XmTEAR_OFF_ENABLED); n++;
  pulldown=XmCreatePulldownMenu(padre,"",args,n);
  n=0;
  etiqueta=XmCreateLabel(pulldown,"Opciones",args,n);
  XtManageChild(etiqueta);
  n=0;
  separador=XmCreateSeparator(pulldown,"",args,n);
  XtManageChild(separador);
  n=0;
  boton=XmCreatePushButton(pulldown,"Carta",args,n);
  XtManageChild(boton);
  n=0;
  boton=XmCreatePushButton(pulldown,"Articulo",args,n);
  XtManageChild(boton);
  n=0;
  boton=XmCreatePushButton(pulldown,"Libro",args,n);
  XtManageChild(boton);
  xmcad=XmStringCreateLtoR("Tipo Documento:",char_set);
  n=0;
  XtSetArg(args[n],XmNlabelString,xmcad); n++;
  XtSetArg(args[n],XmNsubMenuId,pulldown); n++;
  menupane=XmCreateOptionMenu(padre,"menu_option",args,n);
  XtManageChild(menupane);
  XmStringFree(xmcad);
  return menupane;
}

void main(int argc, char *argv[])
{
  Widget menu_bar,menu_popup,menu_option;
  Arg args[16];
  register int n;
  toplevel=XtAppInitialize(&app_context,"ventana_pral",NULL,0,&argc,argv,
NULL,NULL,0);
  n=0;
  XtSetArg(args[n],XmNshowSeparator,True); n++;
  vent_pral=XmCreateMainWindow(toplevel,"vent_pral",args,n);
  XtManageChild(vent_pral);
/* Crear MenuBar */
  menu_bar=crear_menu_bar(vent_pral);
/* Area de trabajo */
  n=0;
  XtSetArg(args[n],XmNwidth,200); n++;
  XtSetArg(args[n],XmNheight,200); n++;
  tablon=XmCreateBulletinBoard(vent_pral,"tablon",args,n);
  XtManageChild(tablon);
  XmMainWindowSetAreas(vent_pral,menu_bar,NULL,NULL,NULL,tablon);
/* Crear menu Poup */
  menu_popup=crear_menu_popup(tablon);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
  XtAddEventHandler(tablon,ButtonPressMask,False,(XtEventHandler)sacar_popup,
(XtPointer)menu_popup);
/* Crear menu Option */
  menu_option=crear_menu_option(tablon);
  n=0;
  XtSetArg(args[n],XmNx,10); n++;
  XtSetArg(args[n],XmNy,10); n++;
  XtSetValues(menu_option,args,n);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

mueve.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 XWindowAttributes win_att;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
/* Atributos de la ventana */
 XGetWindowAttributes(dpy,win,&win_att);
 XSelectInput(dpy,win,ButtonPressMask | KeyPressMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);

 while (1) {
XNextEvent(dpy,&ev);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
switch(ev.type){
   case KeyPress:
XFreeGC(dpy,gc);
        XDestroyWindow(dpy,win);
        XCloseDisplay(dpy);
        exit(1);
break;
  case ButtonPress:
     XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
     switch(ev.xbutton.button){
case Button1:
   win_att.x+=5;
break;
case Button2:
   win_att.x-=5;
break;
case Button3:
   win_att.y+=5;
break;
     }
    XMoveWindow(dpy,win,win_att.x,win_att.y);
  break;
}
}
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

mueve_hija.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win,win2;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin,thewin2;
 XEvent ev;
 XWindowAttributes win_att;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 thewin2.height=100;
 thewin2.width=100;
 win2 = XCreateSimpleWindow(dpy, win,
0, 0, thewin2.width, thewin2.height, 5, fore,back);
 XSetStandardProperties(dpy,win2,"Ventana saludo II","Hola II",None,
argv,argc,&thewin2);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 XSelectInput(dpy,win,ExposureMask | KeyPressMask | ButtonPressMask);
 XSelectInput(dpy,win2,ExposureMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);
 XMapWindow(dpy, win2);
/* Atributos de la ventana */
 XGetWindowAttributes(dpy,win2,&win_att);
 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0){
        if (ev.xexpose.window==win)
  XDrawString(ev.xexpose.display,ev.xexpose.window,
gc,70,95,"Hola Mundo",10);
  else
  XDrawString(ev.xexpose.display,ev.xexpose.window,
gc,30,45,"Hola Mundillo",13);
}
  break;
  case KeyPress:
       XFreeGC(dpy,gc);
       XDestroyWindow(dpy,win);
       XCloseDisplay(dpy);
     exit(1);
  break;
  case ButtonPress:
       XDrawString(ev.xexpose.display,ev.xexpose.window,
gc,70,95,"Hola Mundo",10);
       switch(ev.xbutton.button){
    case Button1:
       win_att.x+=5;
       break;
    case Button2:
       win_att.x-=5;
       break;
    case Button3:
       win_att.y+=5;
       break;
    case Button4:
       win_att.y-=5;
       break;
  }
       XMoveWindow(dpy,win2,win_att.x,win_att.y);
  default:
  break;
}
}
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

mueve_recta.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 XWindowAttributes win_att;
 int x,y;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
/* Atributos de la ventana */
 XGetWindowAttributes(dpy,win,&win_att);
 XSelectInput(dpy,win,ButtonPressMask | KeyPressMask | ButtonMotionMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
   case KeyPress:
XFreeGC(dpy,gc);
        XDestroyWindow(dpy,win);
        XCloseDisplay(dpy);
        exit(1);
break;
  case ButtonPress:
     XDrawLine(ev.xexpose.display,ev.xexpose.window,gc,
0,0,ev.xbutton.x,ev.xbutton.y);
  break;
  case MotionNotify:
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
    switch(ev.xbutton.button){
     case Button1:
      XClearWindow(dpy,win);
     break;
     case Button2:
      XSetFunction(dpy,gc,GXxor);
      XDrawLine(ev.xexpose.display,ev.xexpose.window,gc,0,0,
x,y);
      x=ev.xbutton.x;
      y=ev.xbutton.y;
      XSetFunction(dpy,gc,GXcopy);
     break;
     }
    XDrawLine(ev.xexpose.display,ev.xexpose.window,gc,
0,0,ev.xbutton.x,ev.xbutton.y);
  break;
}
}
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

pido_color.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 XWindowAttributes win_att;
 int x,y;
 XSetWindowAttributes atributos;
 XImage          *imag;
 XColor color,color2;
 Colormap cmap;
 unsigned long col=1;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);
 cmap=DefaultColormap(dpy,scr);
 printf("Celdas de color %d, Profundidad %d\n",XDisplayCells(dpy,scr),
XDisplayPlanes(dpy,scr));
 thewin.height=200;
 thewin.width=200;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 atributos.background_pixel=back;
 atributos.border_pixel=fore;
 atributos.colormap=cmap;

 win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
8,InputOutput,DefaultVisual(dpy,scr),
CWColormap | CWBackPixel | CWBorderPixel, &atributos);

 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);

 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);

/* Atributos de la ventana */
 XGetWindowAttributes(dpy,win,&win_att);

 XSelectInput(dpy,win,ButtonPressMask | KeyPressMask | ExposureMask);

/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);


 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
   case Expose:
                        imag=XGetImage(dpy,win,0,0,thewin.width,thewin.height,
AllPlanes,XYPixmap);
        break;
   case KeyPress:
XFreeGC(dpy,gc);
        XDestroyWindow(dpy,win);
        XCloseDisplay(dpy);
        exit(1);
break;
  case ButtonPress:
    switch(ev.xbutton.button){
     case Button1:
      XPutPixel(imag,ev.xbutton.x,ev.xbutton.y,fore);
      XPutImage(dpy,win,gc,imag,0,0,0,0,thewin.width,
thewin.height);
     break;
     case Button2:
      col*=2;
      if (col>65536) col=1;
      color.red=col-1;
      color.green=0;
      color.blue=0;
      printf("Pedido R %d G %d B %d\n",color.red,
color.green,color.blue);
      if (!XAllocColor(dpy,cmap,&color))
 printf("Error\n");
      XQueryColor(dpy,cmap,&color);
      printf("Obtenido R %d G %d B %d\n",color.red,
color.green,color.blue);
      fore=color.pixel;
     break;
     case Button3:
      XAllocNamedColor(dpy,cmap,"red",&color,&color2);
      XQueryColor(dpy,cmap,&color);
      printf(" Pedido R %d G %d B %d\n",color.red,
color.green,color.blue);
      printf("Obtenido R %d G %d B %d\n",color2.red,
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
color2.green,color2.blue);
      fore=color2.pixel;
     break;

     }
  break;
}
}
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

pixmap-bitmap.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
/* Puede utilizarse para crear bitmaps la aplicaciÆn X bitmap */
/*#include <bitmaps/icon_bitmap>*/

#define dummy_width 40
#define dummy_height 40
 static char bits[] = {
    0x00, 0x80, 0x20, 0x20, 0x20, 0x00, 0x40, 0x40, 0x20, 0x20, 0xc0, 0x41,
    0x40, 0x20, 0x20, 0x30, 0x46, 0x40, 0x20, 0x20, 0x08, 0x88, 0x20, 0x40,
    0x10, 0x08, 0x08, 0x11, 0x80, 0x0f, 0x04, 0x10, 0x0e, 0x00, 0x00, 0x04,
    0x10, 0x00, 0xfe, 0x00, 0x04, 0x10, 0x80, 0x01, 0x03, 0x08, 0x08, 0x40,
    0x00, 0x04, 0x38, 0x08, 0x20, 0x00, 0x08, 0xf6, 0x06, 0x20, 0x1f, 0x08,
    0xc1, 0x03, 0x10, 0x11, 0x10, 0x00, 0x02, 0x10, 0x11, 0x10, 0x00, 0x04,
    0x10, 0xfd, 0x1f, 0x00, 0x04, 0x90, 0x1f, 0xf0, 0x00, 0x08, 0x50, 0x00,
    0x10, 0x00, 0x08, 0x30, 0x00, 0x10, 0x00, 0x08, 0x18, 0x00, 0x10, 0x00,
    0x04, 0x24, 0x00, 0x08, 0x00, 0x04, 0x22, 0x00, 0x08, 0x00, 0x82, 0x4f,
    0x00, 0x04, 0x01, 0x43, 0x91, 0x01, 0x03, 0xc6, 0xa0, 0x20, 0xfe, 0x00,
    0x38, 0x90, 0x40, 0x00, 0x00, 0x00, 0x88, 0x80, 0x00, 0x00, 0x00, 0x48,
    0x80, 0x00, 0x00, 0x00, 0x48, 0x80, 0x00, 0x00, 0x3e, 0x48, 0x80, 0x00,
    0x00, 0x41, 0x48, 0x80, 0x00, 0x00, 0x80, 0x50, 0x40, 0x00, 0x00, 0x80,
    0x60, 0x20, 0x00, 0x00, 0x80, 0x40, 0x10, 0x00, 0x00, 0x80, 0xc0, 0x0f,
    0x00, 0x00, 0x80, 0x40, 0x00, 0x00, 0x00, 0x00, 0x41, 0x40, 0x00, 0x00, 0x00,
    0x3e, 0x40, 0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x80,
    0x00, 0x00, 0x00, 0x00, 0x80, 0x00, 0x00, 0x00};
main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 XSetWindowAttributes atributos;
 Pixmap ima;
 int x,y;
 unsigned int ancho,alto;


 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 ima=XCreatePixmapFromBitmapData(dpy,DefaultRootWindow(dpy), bits, dummy_width,
dummy_height, fore,back,DefaultDepth(dpy,scr));
 atributos.background_pixmap=ima;
 atributos.border_pixel=fore;
 win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
DefaultDepth(dpy,scr),InputOutput,DefaultVisual(dpy,scr),
CWBackPixmap || CWBorderPixel, &atributos);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 XSelectInput(dpy,win,ExposureMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0)
XDrawString(ev.xexpose.display,ev.xexpose.window,
gc,70,95,"Hola Mundo",10);
  break;
}
}
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

pixmap-fondo.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 XSetWindowAttributes atributos;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 atributos.background_pixmap=ParentRelative;
 atributos.border_pixel=fore;
 win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
DefaultDepth(dpy,scr),InputOutput,DefaultVisual(dpy,scr),
CWBackPixmap || CWBorderPixel, &atributos);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 XSelectInput(dpy,win,ExposureMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0)
XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  break;
}
}
 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

pixmap-fondo-None.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 XEvent ev;
 XSetWindowAttributes atributos;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 atributos.background_pixel=back;
 atributos.border_pixel=fore;

 win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
DefaultDepth(dpy,scr),InputOutput,DefaultVisual(dpy,scr),
CWBackPixel || CWBorderPixel, &atributos);

 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);

 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);

 XSelectInput(dpy,win,ExposureMask);

/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0)
XDrawString(ev.xexpose.display,ev.xexpose.window,
gc,70,95,"Hola Mundo",10);

  break;
}
}

 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

profundidad-visual.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 XSizeHints thewin;
 XEvent ev;
 int *prof,i,j,cont,cont2;
 XVisualInfo *listavisual,template;


 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 XSelectInput(dpy,win,ExposureMask | KeyPressMask);
/* Profundidades posibles */
 prof=XListDepths(dpy,scr,&cont);
 for (i=0;i<cont;i++){
     printf("Profundidad %d \n",prof[i]);
     /* Visual posibles */
     template.screen=scr;
     template.depth=prof[i];
     listavisual=XGetVisualInfo(dpy, VisualDepthMask | VisualScreenMask,
&template,&cont2);
     printf("Visuals %d\n",cont2);
     for (j=0;j<cont2;j++)
       {
        switch(listavisual[j].class){
           case PseudoColor:
               printf("PseudoColor \n");
       break;
           case StaticColor:
               printf("StaticColor \n");
       break;
           case DirectColor:
               printf("DirectColor \n");
       break;
           case TrueColor:
               printf("TrueColor \n");
       break;
           case GrayScale:
               printf("GrayScale \n");
       break;
           case StaticGray:
               printf("StaticGray \n");
       break;

    }
   }
  }
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0)
XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  break;
  case KeyPress:
       XFreeGC(dpy,gc);
       XDestroyWindow(dpy,win);
       XCloseDisplay(dpy);
     exit(1);
  break;
  default:
  break;
}
}
}
```

propagate.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win,win2;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;
 XSetWindowAttributes atributos;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=500;
 thewin.width=500;
 atributos.background_pixmap=ParentRelative;
 atributos.border_pixel=fore;

 win = XCreateWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5,
DefaultDepth(dpy,scr),InputOutput,DefaultVisual(dpy,scr),
CWBackPixmap || CWBorderPixel, &atributos);

 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);

 thewin.height=200;
 thewin.width=200;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 atributos.background_pixmap=ParentRelative;
 atributos.border_pixel=fore;

 win2 = XCreateWindow(dpy, win,
0, 0, thewin.width, thewin.height, 2,
DefaultDepth(dpy,scr),InputOutput,DefaultVisual(dpy,scr),
CWBackPixmap || CWBorderPixel, &atributos);

 XSetStandardProperties(dpy,win2,"Ventana saludo","Hola",None,
argv,argc,&thewin);

 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);

 XSelectInput(dpy,win,ButtonPressMask);
 XSelectInput(dpy,win2,ExposureMask | KeyPressMask);

/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);
 XMapWindow(dpy, win2);


 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case KeyPress:
      atributos.do_not_propagate_mask=ButtonPressMask;
    XChangeWindowAttributes(dpy,win2,CWDontPropagate,
                        &atributos);
  break;
  case ButtonPress:
XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
170,365,"Hola Mundo",10);

  break;
  case Expose:
      if(ev.xexpose.count==0){
   XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
}

  break;
}
}

 XFreeGC(dpy,gc);
 XDestroyWindow(dpy,win);
 XCloseDisplay(dpy);
}
```

rectas.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
int scr;
GC gc;
unsigned long fore,back;
XSizeHints thewin;
XEvent ev;
XWindowAttributes win_att;

dpy=XOpenDisplay("");
scr=DefaultScreen(dpy);
fore=WhitePixel(dpy,scr);
back=BlackPixel(dpy,scr);

thewin.height=200;
thewin.width=200;
win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
gc = XCreateGC(dpy, win, 0, NULL);
XSetBackground(dpy,gc,back);
XSetForeground(dpy,gc,fore);
/* Atributos de la ventana */
XGetWindowAttributes(dpy,win,&win_att);
XSelectInput(dpy,win,ButtonPressMask | KeyPressMask);
/* Map the window, wait for the Expose events, and paint */
XMapWindow(dpy, win);

while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
   case KeyPress:
XFreeGC(dpy,gc);
        XDestroyWindow(dpy,win);
        XCloseDisplay(dpy);
        exit(1);
break;
  case ButtonPress:
     XDrawLine(ev.xexpose.display,ev.xexpose.window,
gc,0,0,ev.xbutton.x,ev.xbutton.y);
  break;
}
}
XFreeGC(dpy,gc);
XDestroyWindow(dpy,win);
XCloseDisplay(dpy);
}
```

spin.c

```
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/BulletinB.h>
#include <Xm/SpinB.h>
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
#include <Xm/TextF.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;
Widget toplevel,marco,tablon,rotativo1,texto_rot1,texto_rot2;
XmString meses[12],dias[31];

void main(int, char **);
void crea_meses(void);
void borra_meses(void);
void crea_dias(void);
void borra_dias(void);
void maneja_cambio(Widget, XtPointer, XtPointer);

void crea_meses()
{
  register int i;
  char *mes[]={"Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio",
"Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"};

  for (i=0;i<12;i++) {
    meses[i]=XmStringCreateLtoR(mes[i],char_set);
  }
}

void borra_meses()
{
  register int i;

  for (i=0;i<12;i++) {
    XmStringFree(meses[i]);
  }
}

void crea_dias()
{
  register int i;
  char cadena[5];

  for (i=1;i<=31;i++) {
    sprintf(cadena,"%2d",i);
    dias[i-1]=XmStringCreateLtoR(cadena,char_set);
  }
}

void borra_dias()
{
  register int i;

  for (i=1;i<=31;i++) {
    XmStringFree(dias[i]);
  }
}

void maneja_cambio(Widget w, XtPointer client_data, XtPointer call_data)
{
  XmSpinBoxCallbackStruct *spin_struct = (XmSpinBoxCallbackStruct *)call_data;
  char *cadena;

  XmStringGetLtoR(spin_struct->value,char_set,&cadena);

  printf("Seleccion:%s\n",cadena);
}

void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  char *cadena=NULL;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
toplevel=XtAppInitialize(&app_context,"boton",NULL,0,&argc,argv,NULL,NULL,0);

n=0;
marco=XmCreateFrame(toplevel,"",args,n);
XtManageChild(marco);

n=0;
XtSetArg(args[n],XmNwidth,200); n++;
XtSetArg(args[n],XmNheight,100); n++;
tablon=XmCreateBulletinBoard(marco,"tablon",args,n);
XtManageChild(tablon);

n=0;
XtSetArg(args[n],XmNx,10); n++;
XtSetArg(args[n],XmNy,10); n++;
rotativo1=XmCreateSpinBox(tablon,"rotativo",args,n);

crea_meses();

n=0;
XtSetArg(args[n],XmNwidth,70); n++;
XtSetArg(args[n],XmNvalues,meses); n++;
XtSetArg(args[n],XmNnumValues,12); n++;
XtSetArg(args[n],XmNselectionPolicy,XmSINGLE_SELECT); n++;
XtSetArg(args[n],XmNeditable,False); n++;
texto_rot1=XmCreateTextField(rotativo1,"texto_rot1",args,n);

crea_dias();

n=0;
XtSetArg(args[n],XmNwidth,30); n++;
XtSetArg(args[n],XmNvalues,dias); n++;
XtSetArg(args[n],XmNnumValues,31); n++;
XtSetArg(args[n],XmNselectionPolicy,XmSINGLE_SELECT); n++;
XtSetArg(args[n],XmNeditable,False); n++;
texto_rot2=XmCreateTextField(rotativo1,"texto_rot2",args,n);

XtManageChild(rotativo1);

XtAddCallback(rotativo1,XmNmodifyVerifyCallback,maneja_cambio,NULL);

XtManageChild(texto_rot1);
XtManageChild(texto_rot2);

borra_meses();
borra_dias();

XtRealizeWidget(toplevel);
XtAppMainLoop(app_context);
}
```

tamaño.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
unsigned long fore,back;
XSizeHints thewin;
XEvent ev;
XWindowAttributes win_att;

dpy=XOpenDisplay("");
scr=DefaultScreen(dpy);
fore=WhitePixel(dpy,scr);
back=BlackPixel(dpy,scr);

thewin.height=200;
thewin.width=200;
win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
gc = XCreateGC(dpy, win, 0, NULL);
XSetBackground(dpy,gc,back);
XSetForeground(dpy,gc,fore);
/* Atributos de la ventana */
XGetWindowAttributes(dpy,win,&win_att);
XSelectInput(dpy,win,ExposureMask | ButtonPressMask | KeyPressMask);
/* Map the window, wait for the Expose events, and paint */
XMapWindow(dpy, win);

while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
   case KeyPress:
XFreeGC(dpy,gc);
        XDestroyWindow(dpy,win);
        XCloseDisplay(dpy);
        exit(1);
break;
  case ButtonPress:
     switch(ev.xbutton.button){
case Button1:
   win_att.width+=5;
break;
case Button2:
   win_att.width-=5;
break;
case Button3:
   win_att.height+=5;
break;
     }
    XResizeWindow(dpy,win,win_att.width,win_att.height);
  break;
  case Expose:
    XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  break;
}
}
XFreeGC(dpy,gc);
XDestroyWindow(dpy,win);
XCloseDisplay(dpy);
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

teclas.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/keysym.h>
#include <string.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin;
 XEvent ev;

 int caracter,buf=20;
 char buffer[20],string[20];
 XComposeStatus compo;
 KeySym key;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);

 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);

 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);

 XSelectInput(dpy,win,ExposureMask | KeyPressMask);

/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0)
XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);

  break;
  case KeyPress:
   caracter=XLookupString(&ev,buffer,buf,&key,&compo);
   strcat(string,buffer);
   printf("cadena %s, longitud %d\n",string, caracter);
   if(key==XK_Q){
       XFreeGC(dpy,gc);
       XDestroyWindow(dpy,win);
       XCloseDisplay(dpy);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
      exit(1);
      }
  break;
  default:
  break;
}
}
}
```

vent-deslizante.c

```
#include <Xm/Xm.h>
#include <Xm/Frame.h>
#include <Xm/ScrolledW.h>
#include <Xm/RowColumn.h>
#include <Xm/Label.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,filacol,label[6],ventanad,frame;

void main(int, char **);

void main(int argc, char *argv[])
{
  register int i;
  Arg args[16];
  register int n;
  char *cadena[6]={"etiqueta 1", "etiqueta 2", "etiqueta 3", "etiqueta 4",
"etiqueta 5", "etiqueta 6"};

  toplevel=XtAppInitialize(&app_context,"vent_deslizante",NULL,0,&argc,argv,
NULL,NULL,0);

  n=0;
  frame=XmCreateFrame(toplevel,"frame",args,n);
  XtManageChild(frame);

  n=0;
  XtSetArg(args[n],XmNwidth,100); n++;
  XtSetArg(args[n],XmNheight,100); n++;
  XtSetArg(args[n],XmNscrollingPolicy,XmAUTOMATIC); n++;
  ventanad=XmCreateScrolledWindow(frame,"vent_desl",args,n);
  XtManageChild(ventanad);

  XmScrolledWindowSetAreas(ventanad,NULL,NULL,NULL);

  n=0;
  filacol=XmCreateRowColumn(ventanad,"filacol",args,n);
  XtManageChild(filacol);

  for (i=0;i<6;i++) {
    n=0;
    XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR(cadena[i], char_set));
n++;
    label[i]=XmCreateLabel(filacol,"etiqueta",args,n);
  }
  XtManageChildren(label,6);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

ventana-pral.c

```
#include <Xm/Xm.h>
#include <Xm/MainW.h>
#include <Xm/RowColumn.h>
#include <Xm/CascadeB.h>
#include <Xm/ScrolledW.h>
#include <Xm/PushB.h>
#include <Xm/BulletinB.h>
#include <Xm/Label.h>
#include <Xm/TextF.h>

XtAppContext app_context;
XmStringCharSet char_set=XmSTRING_DEFAULT_CHARSET;

Widget toplevel,vent_pral,area_trabajo;

void main(int, char **);
Widget crea_menu_pral(Widget);
Widget crea_comandos(Widget);
Widget crea_vent_mensaje(Widget);

Widget crea_menu_pral(Widget padre)
{
  Arg args[16];
  register int n;
  Widget menub,menupane,boton,cascada,etiqueta;

  n=0;
  menub=XmCreateMenuBar(padre,"menu",args,n);
  XtManageChild(menub);

  n=0;
  menupane=XmCreatePulldownMenu(menub,"menu",args,n);

  n=0;
  boton=XmCreatePushButton(menupane,"Abrir",args,n);
  XtManageChild(boton);

  n=0;
  boton=XmCreatePushButton(menupane,"Guardar",args,n);
  XtManageChild(boton);

  n=0;
  boton=XmCreatePushButton(menupane,"Salir",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNsubMenuId,menupane); n++;
  cascada=XmCreateCascadeButton(menub,"Fichero",args,n);
  XtManageChild(cascada);

  n=0;
  menupane=XmCreatePulldownMenu(menub,"menu",args,n);

  n=0;
  boton=XmCreatePushButton(menupane,"Ayuda",args,n);
  XtManageChild(boton);

  n=0;
  XtSetArg(args[n],XmNsubMenuId,menupane); n++;
  cascada=XmCreateCascadeButton(menub,"Ayuda",args,n);
  XtManageChild(cascada);

  n=0;
  XtSetArg(args[n],XmNmenuHelpWidget,cascada); n++;
  XtSetValues(menub,args,n);

  return menub;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
}
Widget crea_vent_mensaje(Widget padre)
{
  Arg args[16];
  register int n;
  Widget tablon,etiqueta,texto;

  n=0;
  tablon=XmCreateBulletinBoard(vent_pral,"tablon",args,n);
  XtManageChild(tablon);

  n=0;
  XtSetArg(args[n],XmNx,10); n++;
  XtSetArg(args[n],XmNy,5); n++;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Introducir comando",
char_set)); n++;
  etiqueta=XmCreateLabel(tablon,"etiqueta",args,n);
  XtManageChild(etiqueta);

  n=0;
  XtSetArg(args[n],XmNx,10); n++;
  XtSetArg(args[n],XmNy,35); n++;
  XtSetArg(args[n],XmNcolumns,50); n++;
  texto=XmCreateTextField(tablon,"etiqueta",args,n);
  XtManageChild(texto);

  return tablon;
}
Widget crea_comandos(Widget padre)
{
  Arg args[16];
  register int n;
  Widget tablon,etiqueta;

  n=0;
  tablon=XmCreateBulletinBoard(vent_pral,"tablon",args,n);
  XtManageChild(tablon);

  n=0;
  XtSetArg(args[n],XmNx,10); n++;
  XtSetArg(args[n],XmNy,10); n++;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("Ventana de Mensajes",
char_set)); n++;
  etiqueta=XmCreateLabel(tablon,"etiqueta",args,n);
  XtManageChild(etiqueta);

  return tablon;
}
void main(int argc, char *argv[])
{
  Arg args[16];
  register int n;
  Widget menu,etiqueta,comando,mensaje;

  toplevel=XtAppInitialize(&app_context,"ventana_pral",NULL,0,&argc,argv,
NULL,NULL,0);

  n=0;
  XtSetArg(args[n],XmNshowSeparator,True); n++;
  vent_pral=XmCreateMainWindow(toplevel,"vent_pral",args,n);
  XtManageChild(vent_pral);

/* Creacion de la barra de menu principal */

  menu=crea_menu_pral(vent_pral);
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
/* Creacion de la ventana de comandos */
  comando=crea_comandos(vent_pral);
/* Area de trabajo */
  n=0;
  XtSetArg(args[n],XmNwidth,100); n++;
  XtSetArg(args[n],XmNheight,100); n++;
  area_trabajo=XmCreateBulletinBoard(vent_pral,"tablon",args,n);
  XtManageChild(area_trabajo);
  n=0;
  XtSetArg(args[n],XmNlabelString,XmStringCreateLtoR("AREA DE TRABAJO",
char_set)); n++;
  etiqueta=XmCreateLabel(area_trabajo,"etiqueta",args,n);
  XtManageChild(etiqueta);
  XmMainWindowSetAreas(vent_pral,menu,comando,NULL,NULL,area_trabajo);
/* Ventana de mensajes */
  mensaje=crea_vent_mensaje(vent_pral);
  n=0;
  XtSetArg(args[n],XmNmessageWindow,mensaje); n++;
  XtSetValues(vent_pral,args,n);
  XtRealizeWidget(toplevel);
  XtAppMainLoop(app_context);
}
```

ventana-hermanas.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win,win2;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin,thewin2;
 XEvent ev;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 thewin2.height=100;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 thewin2.width=100;
 win2 = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin2.width, thewin2.height, 5, fore,back);
 XSetStandardProperties(dpy,win2,"Ventana saludo II","Hola II",None,
argv,argc,&thewin2);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 XSelectInput(dpy,win,ExposureMask | KeyPressMask);
 XSelectInput(dpy,win2,ExposureMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);
 XMapWindow(dpy, win2);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0){
        if (ev.xexpose.window==win)
  XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  else
  XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
30,45,"Hola Mundillo",13);
}
  break;
  case KeyPress:
       XFreeGC(dpy,gc);
       XDestroyWindow(dpy,win);
       XCloseDisplay(dpy);
     exit(1);
  break;
  default:
  break;
}
}
}
```

ventanas-padre-hija.c

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

main(argc, argv)
int argc;
char **argv;
{
 Display *dpy;
 Window win,win2;
 int scr;
 GC gc;
 unsigned long fore,back;
 XSizeHints thewin,thewin2;
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

```
 XEvent ev;

 dpy=XOpenDisplay("");
 scr=DefaultScreen(dpy);
 fore=WhitePixel(dpy,scr);
 back=BlackPixel(dpy,scr);

 thewin.height=200;
 thewin.width=200;
 win = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy),
0, 0, thewin.width, thewin.height, 5, fore,back);
 XSetStandardProperties(dpy,win,"Ventana saludo","Hola",None,
argv,argc,&thewin);
 thewin2.height=100;
 thewin2.width=100;
 win2 = XCreateSimpleWindow(dpy, win,
0, 0, thewin2.width, thewin2.height, 5, fore,back);
 XSetStandardProperties(dpy,win2,"Ventana saludo II","Hola II",None,
argv,argc,&thewin2);
 gc = XCreateGC(dpy, win, 0, NULL);
 XSetBackground(dpy,gc,back);
 XSetForeground(dpy,gc,fore);
 XSelectInput(dpy,win,ExposureMask | KeyPressMask);
 XSelectInput(dpy,win2,ExposureMask);
/* Map the window, wait for the Expose events, and paint */
 XMapWindow(dpy, win);
 XMapWindow(dpy, win2);

 while (1) {
XNextEvent(dpy,&ev);
switch(ev.type){
  case Expose:
      if(ev.xexpose.count==0){
        if (ev.xexpose.window==win)
  XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
70,95,"Hola Mundo",10);
  else
  XDrawString(ev.xexpose.display,ev.xexpose.window,gc,
30,45,"Hola Mundillo",13);
}
  break;
  case KeyPress:
       XFreeGC(dpy,gc);
       XDestroyWindow(dpy,win);
       XCloseDisplay(dpy);
     exit(1);
  break;
  default:
  break;
}
}
}
```

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005

Bibliografía

[1] Using the X Window System. Hewlett-Packard, 1991.

[2] X Window Course. D. Marshall, 1994.

[3] X Window System User's Guide. O'Reilly & Associates, Inc., 1989.

[4] L. Reiss, J. Radin: Aplique X Window. Osborne McGraw-Hill, 1993.

[5] Adrian Nye: Xlib Programmming Manual. O'Reilly & Associates, Inc., 1992.

[6] Xlib Reference Manual. O'Reilly & Associates, Inc., 1992.

[7] OSF/Motif Programmer's Guide Revision 2.0 Open Software Foundation Inc., 1994.

[8] Donald L. McMinds: Mastering OSF/Motif Widgets 2nd Edition. Addison-Wesley Publishing Company Inc., 1993.

[9] Open Software Foundation: OSF/Motif Programmer's Reference Revision 1.2. PTR Prentice-Hall, Englewood Cliffs, 1992.

[10] Jan D. Newmarch: The X Window System and Motif. A Fast Track Approach. Addison-Wesley Pub. Company, 1992.

[11] X Toolkit Intrinsics Reference Manual. Third Edition. O'Reilly & Associates, Inc., 1992.

[12] Open Software Foundation: OSF/Motif Style Guide. PTR Prentice-Hall, Englewood Cliffs, 1991.

[13] Adrian Nye and Tim O'Reilly: X Toolkit Intrinsics Programming Manual. O'Reilly & Associates, Inc., 1992.

[14] Paula M. Ferguson: Motif Reference Manual. O'Reilly & Associates, Inc., 1994.

[15] Dan Heller & Paula M. Ferguson:Motif Programming Manual. Second Edition. O'Reilly & Associates, Inc., 1994.

© Del documento, los autores. Digitalización realizada por ULPGC. Biblioteca Universitaria, 2005