UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Controles para el desarrollo de interfaces en *MetaCard*

Apellidos, nombre	Agustí i Melchor, Manuel (magusti@disca.upv.es)
Departamento	Departamento de Informática de Sistemas y Computadores
Centro	Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Resumen

En este artículo y desde la perspectiva de una herramienta de autor multiplataforma como es MetaCard *[1]*, se va a explorar el uso de los elementos auxiliares del interfaz de aplicaciones en un entorno de ventanas: las barras de desplazamiento, las ventanas y la propia aplicación.

2 Objetivos

La idea de construcción de una aplicación en *MetaCard* pasa por una primera parte dedicada al desarrollo del interfaz utilizando una aproximación visual: creación de ventanas y disposición de elementos en las mismas sobre los que el usuario interactúa. En este caso nos centraremos en:

- El interfaz de la aplicación como una ventana, en un sistema gráfico basado en la metáfora tradicional del escritorio, que nos llevará a explorar los conceptos de pilas (*stacks*), subpilas y tarjetas (*cards*).
- Exponer los tipos y operaciones a realizar sobre los controles de tipo barra de desplazamiento (*scrollbar*).

En ambos casos se acompañará de ejemplos, para facilitar la experimentación, por parte del lector interesado, de lo que aquí se expondrá.

3 Barras de desplazamiento (*ScrollBar*)

El objeto "scrollbar" permite elaborar controles en los que tenga sentido este concepto (la fig. 1 muestra un ejemplo de esta variedad), así como otros más complejos (grupos) en los cuales haya que proporcionar un mecanismo de desplazamiento del contenido visible de los mismos en el espacio que para ellos se haya dedicado en una ventana.

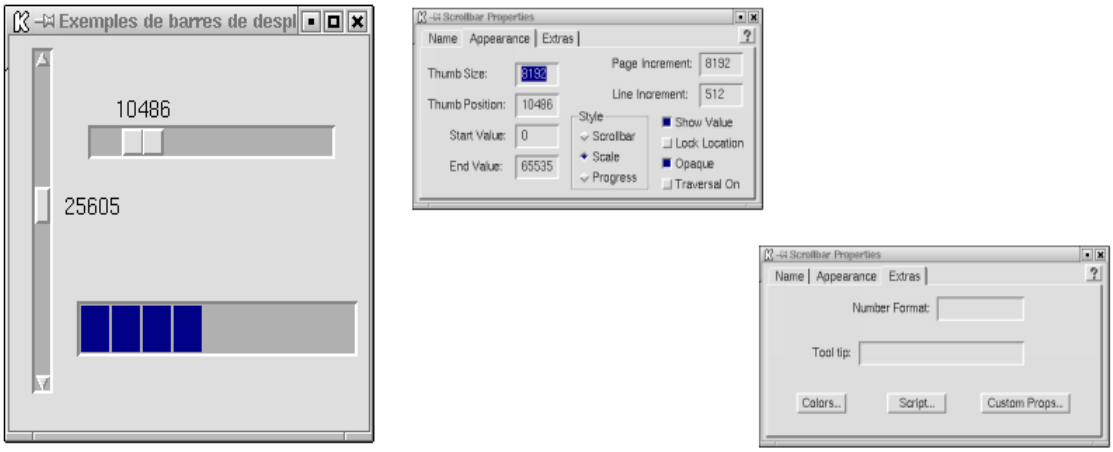

Figura 1: Aspecto y propiedades de un objeto tipo "scrollbar" (barra de desplazamiento).

Como ejemplo de este objeto y de su utilización, a continuación le muestro cómo he implementado una barra de estado en la que se ofrece una realimentación visual del estado de progreso de una serie de experimentos cuya duración aconsejaba dar muestras de la evolución de la misma y no dejar al usuario con la duda de "en qué punto se encontrará la ejecución del programa". Además lo utilicé como excusa para guardar el estado en cada ronda, de manera que si el programa o el sistema se tenía que reiniciar, se podía reengachar la ejecución del programa desde un punto de sincronismo cercano que evitase repetir toda la secuencia de experimentos desde su inicio.

La barra de progreso (“barraEstat”) se puede ver en la parte de arriba de la fig. 2. Forma parte, junto con un campo de texto (field “lineaEstat”), de la “barra de mensajes” de una aplicación, habitualmente situada en la parte inferior de la ventana. No solo la barra de desplazamiento si no también el cambio de icono del cursor y los mensajes de texto le dicen al usuario qué está haciendo la aplicación.

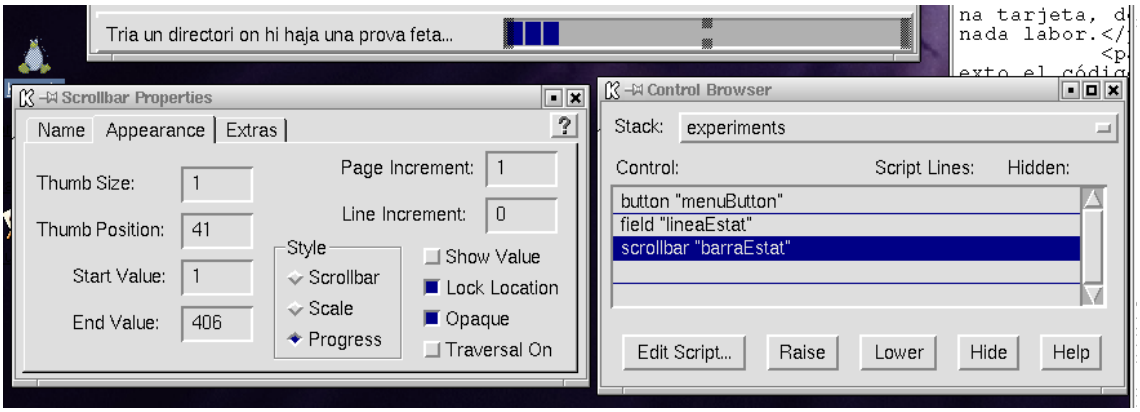

Figura 2: Ejemplo de barra de desplazamiento incorporada a un elemento del interfaz de una aplicación y utilizada para mostrar una representación gráfica de la evolución de un proceso secuencial.

Otros elementos del interfaz no se muestran por brevedad en la fig. 2, pero sí que se citan en el listado 1. Por ejemplo, cada experimento se realiza enviando un mensaje a un botón (*button* “explorarImatgens”), o la mención a un campo de texto que muestra en pantalla la información del directorio de trabajo (field "directoriDeTreball") .

El objetivo de este interfaz es mantener la información en archivos en disco, para poder describir un experimento a realizar en varios pasos o tandas y continuar desde el punto donde se interrumpan, si es el caso. Para ello se apoya en dos ficheros en disco: “llistaExperiments.txt" es lo que se quiere probar en cada tanda, "llistaTandesFetes.txt" contiene los que se han completado, del total que guarda "llistaTandes.txt"

```
# Código del gráfico "grafic"
global timerID
on iteracionsBucleReanudarExperiments parametre sufixe numExperimentsFets numExperiments
  set cursor to busy
  # 1.- Comprobar si debe entrar
  if (numExperimentsFets > numExperiments)
  then
    put line ((the number of lines of URL( "file:" & fld "directoriDeTreball" & \
        "/" & "llistaTandesFetes.txt")) \
        +1) of URL( "file:" & fld "directoriDeTreball" & "/" & "llistaTandes.txt")  & return \
        after URL( "file:" & fld "directoriDeTreball" & "/" & "llistaTandesFetes.txt")
    send "mouseUp" to button "reanudarTandes" in 10 milliseconds
  else
    # 2.- Anunciar el punto actual
    set the thumbPos of scrollBar "barraEstat" of group "grupBarraEstat" to\
          numExperimentsFets
    # 3.- Ejecutar experimento y ...
    send  ("explorarImatgens" &&\
       quote & (line (numExperimentsFets + 1) of\
       URL( "file:" &  fld "directoriDeTreball" & "/" & "llistaExperiments.txt" )) & quote  & "," &\
       (fld "directoriDeTreball" & "/imatgensNormalitzaes") & "," & \
       (fld "directoriDeTreball" & "/descripcioWavelets") & "," & \
       line ((the number of lines of\
       URL( "file:" & fld "directoriDeTreball" & "/" & "llistaTandesFetes.txt")) \
       + 1) of\
       URL( "file:" & fld "directoriDeTreball" & "/" & "llistaTandes.txt") ) to\
       button  "explorarImatgens"
    # …. y tomar nota si se completa
    put line numExperimentsFets of URL( "file:" &  fld "directoriDeTreball" & "/" & "llistaExperiments.txt" ) \
       & return after URL( "file:" &  fld "directoriDeTreball" & "/" & "llistaExperimentsFets.txt" )
    #4.-  Preparar la siguiente iteración, permitiendo que se interrumpa la secuencia
    send ("iteracionsBucleReanudarExperiments" && parametre & "," & sufixe & "," &\
       (numExperimentsFets + 1) & "," & numExperiments)  to me in 10 milliseconds
    put the result into timerID
  end if
end iteracionsBucleReanudarExperiments
```

Listado 1: Código de ejemplo de barra de desplazamiento.

El código que se utiliza es el que muestra el listado 1 y responde al algoritmo siguiente:

1. Comprobar si debe entrar: Iniciar una tanda si no se ha completado el número de estas.
2. Anunciar con la barra de desplazamiento en modo "progreso" el avance de las tandas: ¿en qué punto estamos?.
3. Dentro de cada tanda se han de ejecutar un número constante de experimentos y guardar el experimento que se acaba de completar.
4. Preparar la siguiente iteración y atender a que el usuario puede querer interrumpir el proceso. Terminado un experimento al menos, de una tanda, se puede hacer y continuar desde aquí, posteriormente, es posible tomando nota de donde se ha completado.

4 Tarjetas (*card*)

Son los contenedores de cada posible "vista" de una pila. Aquí se disponen los objetos con los que el usuario interactúa, la única diferencia con los restantes objetos es que no se puede mover y que su tamaño siempre está regido por el de la pila. Pero sigue siendo un objeto de *MetaCard* , por lo que es posible:

- Incluir en su código asociado que maneje los habituales eventos que puede recibir cualquier objeto, si es que le llegan en la jerarquía de paso de mensajes, como por ejemplo un **mouseUp** o cualquiera que se invente el desarrollador.
- Decorarse con colores, patrones e, incluso, con una imagen.

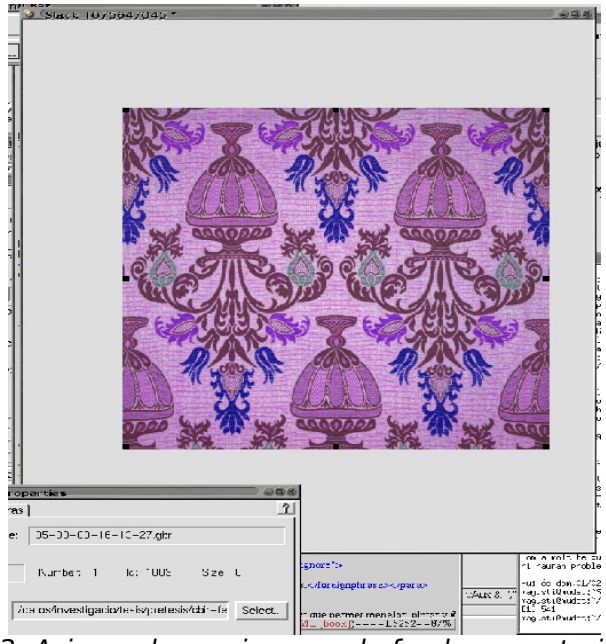

Figura 3: Asignando una imagen de fondo a una tarjeta.

Desde el punto de vista de este documento, interesa en tanto en cuanto es un posible contenedor de información gráfica. El motivo puede ser estético, pero también informativo. En cualquier caso podemos estar interesado en mostrar una información, de tipo imagen en mapa de bits, de fondo que sea diferente en cada tarjeta. Para ello, los materiales a emplear son cualquier formato soportado por *MetaCard*, que deberá ser importado en la pila y hecho no visible. Para estar disponible en la pila y tener un identificador asignado a través del cual la referiremos

Se hace efectiva esta asignación mediante la propiedad **backgroundPattern** que se utiliza, de forma abreviada, en el listado 2, como una orden que se ha ejecutado en la "*Message Box*" y cuyo resultado se puede ver en la fig. 3. Por cierto, cuando quiera se puede cambiar, e incluso eliminarlo con otra orden mostrada en el mismo listado. La imagen se puede ajustar y replicar para llenar todo el espacio disponible.

```
#Asignar
set the backPattern of this card to "1003"


# Eliminar la asignación
set the backPattern of this card to ""
```

Listado 2: Código de ejemplo de asignación de una imagen como fondo de una tarjeta.

Y, un último aspecto de esta utilización de imágenes a tarjetas: en todos los sistemas operativos con soporte para interfaces gráficas es posible utilizar ventanas de aspecto no rectangular. Esto se puede hacer si se dispone de imágenes con colores transparentes. Puede así diseñar y utilizar, si es necesario, el aspecto de su aplicación mediante el programa de creación y edición de imágenes de su preferencia y utilizar las características de transparencia de los formatos que la soportan. La fig. 4 muestra un ejemplo realizado con *The Gimp* para la creación de una imagen en un formato libre. Asumiendo que el color trasparente es el que rellena esta especie de “roto” hasta formar un rectángulo, la ventana que se mostraría en pantalla tomaría la geometría de esa área puntiaguda de la figura.

Figura 4: Aspecto no regular de una ventana realizado con una imagen PNG con fondo transparente mediante The Gimp.

El lector, seguro, ya se habrá percatado que puede navegar entre las tarjetas de una pila. de una tarjeta. Veamos cómo responder los los eventos **preOpenCard** y **openCard** que se generan en estas situaciones. El **preOpenCard**, se ejecutará en un momento en que todavía "no está" en la tarjeta, aunque ya puede actuar sobre sus objetos, mientras que con el **openCard** ya está en la misma y los objetos que contiene son visibles (los que tengan esta propiedad activa, claro está) .

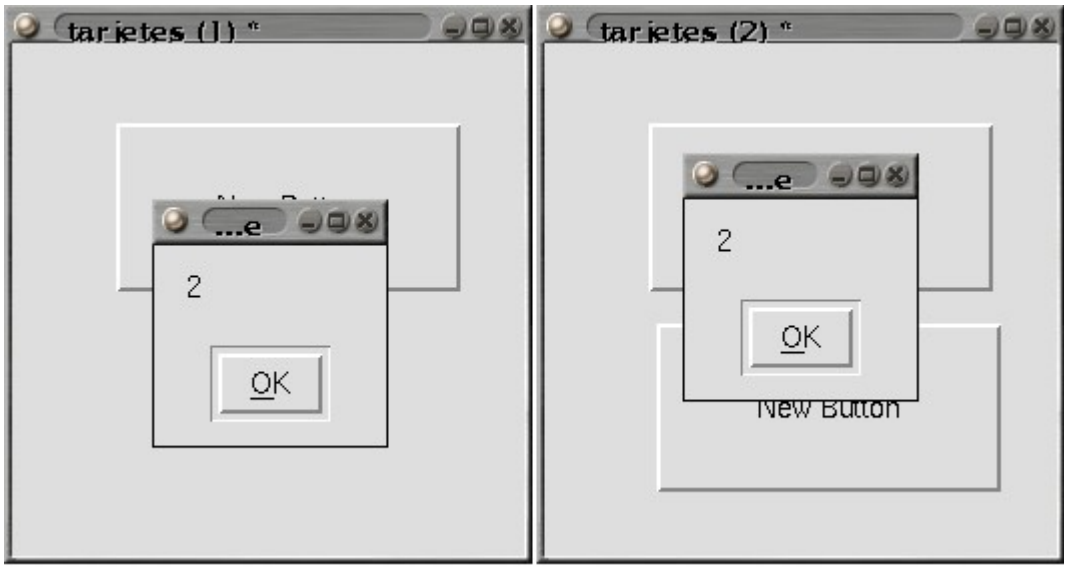

Figura 5: Eventos preOpenCard y openCard en tarjetas.

```
# script de las tarjetas
on preOpenCard
  answer the number of controls of me titled ("preOpenCard" && "the short name of me")
end preOpenCard


on openCard
  answer the number of controls of me titled ("openCard" && "the short name of me")
end openCard
```

Listado 3: Código de ejemplo de asignación de una imagen como fondo de una tarjeta.

Para ilustrar este punto construyamos una pila con dos tarjetas y un número diferente de objetos en cada una, para que sea más fácil identificarla. La fig. 5: e muestra una pila básica, mostrando el contenido de las dos únicas tarjetas que la forman para mostrar la diferencia entre los eventos **preOpenCard** y **openCard**. A cada tarjeta le asociaremos un código que muestre en pantalla cuántos objetos tiene y comprobaremos visualmente cuándo suceden. El único

código necesario es el que muestra el listado 3 y que ha de estar en ambas tarjetas:

Al utilizar las flechas del cursor (que es el modo por defecto, si no ha sido desactivado) para hacer el tránsito de una tarjeta a la siguiente, se observarán los dos estados que se muestran en la siguiente figura. El resultado del avance desde la tarjeta 1 a la 2 se realiza en dos partes: primero se ejecuta el código asociado al evento **preOpenCard** (imagen de la izquierda) y después al **openCard**, mostrado en la imagen d e la derecha.

5 Pilas (stack) y subpilas

En este texto utilizamos indistintamente el término pila y aplicación, puesto que es contenedor de todos los demás, así que las acciones sobre ellas repercutirán en todo lo que contienen. Cada ventana que aparece en pantalla es una pila, desde el punto de vista de *MetaCard*.

Ahora haremos un breve repaso a las operaciones típicas sobre ellas y lo acompañaremos con código que puede escribir y ejecutar. ¿Qué cosas se pueden hacer con una pila? Lo que se le ocurra, hagámoslo simplemente desde la "*Message Box*" como se muestran en el listado 4.

```
# abrir una nueva pila
topLevel "MC stack Properties Menu"


# Propiedades de una pila
put the width of stack "Home"
set the width of stack "Home" to 500
set the height of stack "Home" to 344
set the left of stack "Home" to 344
# Otras operaciones
hide stack "Home"
go stack "Home"
move stack "Home" to "10,10"
close stack "Home"
```

Listado 4: Ejemplos de órdenes relativas al manejo de pilas (*stacks*).

¿Reconoce la pila que se ha abierto al ejecutar la primera orden? Algunas pueden estar subordinadas a otras y por ello distinguimos entre pilas y subpilas. Puede utilizar respectivamente **topLevel**, para abrir una nueva pila. Para abrir una subpila se puede utilizar **go**, y también para transitar entre las tarjetas de una pila; pero tiene muchas más opciones (la ayuda en línea le mostrará como abrir una pila sin dibujarla en pantalla, o como ir a una tarjeta determinada de otra pila, etc).

Es lo más habitual, pero recuerde que es un objeto más de los que podemos utilizar, así que se puede averiguar y cambiar las propiedades usuales: color, geometría, texto, nombre, ...: También ocultarla o mostrarla, cerrarla, ... y, claro, moverla por la pantalla. Como cualquier otro objeto, puede responder a los eventos del sistema o a los que defina el programador.

Por brevedad sólo citaré que existen otras para gobernar el modo en que abre una pila: esto es si se cede o no el foco, si el usuario puede pasar a trabajar con otra pila o ha de responder (o esperar) a lo que una le dice (o le pregunta) antes de continuar. La ayuda, como siempre, le asiste a la hora de recordarlas y le explica las diferencias entre **model, modeless, open** y **palette**. Para cerrar una pila ..., ¡correcto! **close stack**, aunque también existen otras posibilidades como **quit**. Y entre las características más estéticas de una pila hay que mencionar **decorations** y **backdrop**.

6 Tutorial sobre tarjetas y pilas

Pongámoslo todo junto y vamos a crear un "tutorial" sobre el manejo y la actuación con una pila. El esquema general es un botón que desencadena la acción oportuna y su código que es mostrado en un campo de texto.

Cada tarjeta contiene esta pareja y es un único caso de aplicación. Para que aparezca el código en el campo de texto cada vez que cambiamos de tarjeta, todas las tarjetas (asumiendo los nombre de "accio" para el botón y "scriptAccio" para el campo de texto) tienen este mismo código, véase listado 5. Cada tarjeta es un caso de aplicación y le propongo estas dos acciones, para las que puede consultar una solución en el mencionado listado 5:

- Mover la pila por pantalla a una posición aleatoria dento de los márgenes del escritorio.
- Modificar, aleatoriamente, las dimensiones de la ventana.

7 Conclusiones

A lo largo de este documento hemos visto el uso de las operaciones típicas sobre los controles de interfaz más complejos en el uso de *MetaCard*: las barras de desplazamiento, las tarjetas y las pilas.

La exposición de sus tipos y modos de operación se ha acompañado de ejemplos breves, para que el lector interesado haya podido experimentar con ellos.

```
# Caso de estudio: tutorial  sobre manejo  de operaciones
# Cualquier tarjeta
on opernCard
 put the script of buttin “accio” into fld “scriptAccio”
end  opernCard

#button "accio", etiqueta "Mover la pila por  la pantalla"
on mouseUp
  move this stack to random(the third item of the screenRect), \
                  random(the fourth item of the screenRect)
end mouseUp

#button "accio", etiqueta "Redimensionar"
global ampleActual, altActual
on mouseUp
  # put "Tamany de la pantalla" && the screenRect &&\
     # "tamany de la pila" && the width of this stack &\
           # "x" &\
     # the height of this stack
  if (the width of this stack = ampleActual)
  then
    set the width of this stack to random( ampleActual + 100)
    set the height of this stack to random( altActual + 100)
  else
    set the width of this stack to ampleActual
    set the height of this stack to altActual
  end if
end mouseUp
```

Listado 5: Código para un tutorial básico de operaciones sobre pilas y tarjetas.

8 Bibliografía

[1] MetaCard <http://www.metacard.com/>.