Ampliación de Lógica

Programando la interfaz gráfica con:

XPCE/Prolog

Pau Sánchez Campello
4rto Ingenieria en Informatica.

1.- Cargando la librería PCE

Para poder usar predicados para trabajar con gráficos bajo *Prolog* deberemos cargar la librería PCE. Para cargar dicha librería pondremos la siguiente linea en cualquier parte del archivo (preferiblemente al principio del mismo):

```
:- use_module(library(pce)).
```

Esta línea lo que hace es decirle a Prolog que cargue la librería una vez ha terminado de compilar, antes de que desde el prompt de Prolog se nos deje hacer ninguna consulta.

Una vez cargada esta librería, ya disponemos de una serie de predicados para poder crear ventanas, botones, ... y una gran variedad de objetos.

2.- Creando objetos e interactuando con ellos

Con la librería PCE se trabaja con un esquema orientado a objetos, donde podremos crear clases y trabajar con diferentes objetos, pudiendo llamar a métodos de dichos objetos (pasándoles los correspondientes parámetros) o llamar a métodos que nos devuelvan algún valor, y obviamente si creamos objetos, también podremos destruirlos.

Así pues principalmente existen 4 predicados con los que se puede trabajar con XPCE/Prolog. Estos predicados sirven para crear objetos, enviar mensajes a objetos, recibir mensajes de objetos y liberar la memoria de los objetos. Estos 4 predicados son:

- ***new(?Reference, +NewTerm)***: Este predicado recoge dos parámetros, el primero recogería la referencia que se le asigna al nuevo objeto, ya que **new** se usa para crear objetos. El segundo parámetro le indicaría el objeto que se quiere crear.

- ***send(?Receiver, +Selector(...Args...))***: El primer parámetro del predicado es una referencia al objeto al que deseemos enviarle un mensaje. El segundo parámetro indicará el método al que queremos invocar, lo cual indicaremos junto a los argumentos que queremos enviarle al método.

- ***get(?Receiver, +Selector(+Argument...), -Result)***: Los dos primeros parámetros tienen el mismo significado que para *send*, sin embargo el último parámetro sirve para recoger el valor que nos devuelva el método que hallamos invocado.

- ***free(?Reference)***: libera la memoria asociada al objeto que se le indica en el primer parámetro.

Las referencias se usan para saber a que objeto nos referimos, así pues cada objeto que creemos deberá tener su propia referencia, ya que después toda memoria que reservemos con **new**, será conveniente liberarla con **free**.

Prolog usa principalmente dos tipos de referencias, una que seria mediante las variables típicas de prolog (una cadena que empiece por mayúscula, como *Variable, Pepe, ...)*, y la otra forma es definir referencias con nombre, las cuales una vez definidas no podremos volver a crear otra referencia para dicho nombre, ya que nos dará error en ejecución. Estas últimas referencias resultan interesantes para acceder de forma global a un mismo objeto, sin tener que pasar ningún tipo de parámetro. Estas variables se crean usando el operador especial **@** así pues, cualquier nombre que empiece por **@** será una variable asociada a ese nombre que le demos (por ejemplo, @pepe, @variable).

Cuando usemos las variables con nombre deberemos llevar especial cuidado ya que deberemos liberarlas usando free antes de que finalice la evaluación del predicado, ya que si volvemos a realizar la misma consulta, y volvemos a crear objetos usando esas mismas variables, dará error y no dejará ejecutar.

Por ejemplo, para crear un dialogo que contenga un botón y que al pulsar sobre el mismo cierre esa ventana:

```
ejemplo :-
        /*
         * Crea el objeto dialogo en la variable D
         */
        new(D, dialog('Nombre del Dialogo')),

        /*
         * Crea el objeto boton almacenandolo en la variable @boton de tal forma
         * que al pulsar sobre el boton libere la memoria y cierre la ventana)
         */
        new(@boton, button('Cerrar Dialogo',
                and(
                        message(D, destroy),
                        message(D, free),
                        message(@boton, free)))),

        /*
         * Inserta el botón en el diálogo
         */
        send(D, append(@boton)),

        /*
         * Le envia el mensaje open al dialogo para que cree y muestre la ventana.
         */
        send(D, open).
```

En este ejemplo podemos ver como se usan los predicados *new, free y send*, así como podemos ver que aparecen otros como *append, open* y *quit*, que son métodos de la clase *dialog* (de la cual hemos creado una instancia en la variable D usando el predicado *new*).

Cuando pasamos parámetros en la creación o en la llamada a un método de un objeto, estos métodos pueden tener dos tipos de parámetros, pueden tener parámetros obligatorios y parámetros opcionales. La diferencia está en que si no indicamos explícitamente un parámetro obligatorio XPCE generará un error, mientras que si no definimos un parámetro opcional, entonces XPCE pondrá en su lugar la constante ***@default*** cuyo valor ya es el que tenga definido por defecto ese método.

3.- Enviando mensajes

Como hemos visto en el ejemplo, determinados objetos pueden ejecutar determinadas acciones, como puede ser el caso del objeto ***button,*** así pues, cuando pulsamos en el botón, podemos hacer que se ejecuten una o mas acciones, en este caso que se evaluen uno o más predicados.

Si únicamente queremos evaluar 1 predicado, podemos hacer uso del predicado:

message(*receiver=object|function, selector=name|function, argument=any|function* ...)

Como se puede observar, el primer parámetro indica quien es el receptor, en cuyo caso, en el ejemplo anterior será siempre un objeto que previamente hemos creado con new. El selector (el segundo parámetro), indicaría el nombre del método o la función o el predicado al que se desea llamar, y los posteriores parámetros son los argumentos de esa función.

Como nota importante cabe destacar que en esos parámetros únicamente pueden ir objetos declarados en el XPCE, es decir, que no podemos poner una lista como tercer o cuarto parámetro y esperar que se le pase esa lista a la función cuando sea llamada (al menos en principio).

En el campo del receptor básicamente podemos especificar una variable que hallamos inicializado con *new*, teniendo entonces que invocar necesariamente a un método de ese objeto, o por el contrario, también se permite invocar a un predicado con sus correspondientes parámetros (exceptuando listas y tipos no básicos que implemente el XPCE), pero en este caso en el campo de receptor deberemos poner **@prolog** y en el segundo parámetro el nombre del predicado y seguir rellenando con los argumentos.

Con ánimo de mostrar y entender mejor como funciona se puede ver el siguiente ejemplo:

```
ejemplo_mensajes :-
        % Crea el objeto dialogo en la variable D
        new(D, dialog('Nombre del Dialogo')),

        % Crea un boton que llama al predicado mostrar_mensaje
        new(B, button('Mostrar en Consola',
                message(@prolog, mostrar_mensaje, 'Este es el valor que tendra la variable P'))),

        % Crea un boton para cerrar el dialogo
        new(@boton, button('Cerrar Dialogo',
                and(
                        message(D, destroy),
                        message(D, free),
                        message(D, free),
                        message(@boton, free)))),

         % Inserta los botones en el diálogo
        send(D, append(@boton)),
        send(D, append(B)),

        % Muestre la ventana.
        send(D, open).

% Muestra un mensaje en la consola.
mostrar_mensaje(P) :-
        write('La variable P vale '), write(P), nl.
```

4.- Creando elementos en el entorno grafico

Una vez explicados los conceptos básicos ya podemos crear objetos en el entorno gráfico, tales como diálogos, textos, etiquetas, recuadros para dibujar, botones, etc... además seguramente en este apartado se entenderá mejor el uso de todo lo anterior.

Como hemos dicho anteriormente, podemos crear varios tipos de objetos, a continuación enumeraré algunos de estos objetos y algunos de sus métodos.

OBJETO	DESCRIPCION
dialog	Sirve para crear un cuadro de dialogo
button	Sirve para crear un boton
cursor	Sirve para modificar la imagen del cursor
figure	Sirve para poner imágenes en pantalla
image	Sirve para cargar imágenes
bitmap	Para convertir una imagen en un elemento gráfico (basicamente hace de puente para pasar de *image* a *figure)*
pixmap	Es practicamente equivalente a image
label	Para escribir una etiqueta por pantalla (puede servir para mostrar un texto)
menu	Para crear un menu
menu_bar	Para crear una barra de menus
menu_item	Para embeder elementos dentro de un menu
point	Para crear un punto con 2 cordenadas
popup	Para crear un popup en una ventana.
slider	Para crear un desplazador
window	Para crear una ventana donde se pueden dibujar otra serie de objetos, como imágenes, etc...

De las clases citadas, únicamente describiré el constructor y algunos métodos de algunas de ellas, y como usarlas o interaccionar con las demás clases.

DIALOG

Esta es la clase básica para crear dialogos.

Constructor:

dialog(*label=[name], size=[size], display=[display]*)

name: indica el titulo a la ventana

size: es de tipo size y sirve para indicar el tamaño de la ventana

display: indica donde queremos que se visualize (mejor no tocar este parametro si no sabemos que estamos haciendo)

Podemos observar que todos los parametros son opcionales, aunque siempre es bueno ponerle un titulo a la ventana

Así, como ejemplo crearemos un dialogo con titulo 'Titulo del dialogo' y de tamaño 440 x 320.

```
new(D, dialog('Titulo del Dialogo', size(440, 320)) ),
```

Métodos:

Esta clase dispone de varios métodos que pueden resultar interesantes, entre ellos tenemos:

append(Objeto): Insertaria el objeto Objeto dentro del dialogo, visualizandolo en el mismo, por ejemplo sirve para insertar un boton o cualquier otra cosa, como en el siguiente ejemplo:

```
send(D, append(button('Boton 1')))
```

open(): abre la ventana de diálogo visualizandola en pantalla:

```
send(D, open),
```

destroy(): cierra la ventana de diálogo visualizada en pantalla:

```
send(D, destroy),
```

BUTTON

Esta es la clase básica para crear botones

Constructor:

button(*name=name, message=[code]*, label=[name]*)

name: indica el nombre del boton (si no se especifica la etiqueta que queramos que tenga el boton, entonces adoptará una etiqueta con el mismo texto que name)

message: indica el mensaje o acción que queremos que sea ejecutado cuando pulsemos sobre el botón con el ratón.

label: Indica la etiqueta que queremos que se muestre en el boton.

```
new(Boton, button('Salir', message(Dialogo, quit)))
```

LABEL

Esta es la clase básica para crear etiquetas de texto

Constructor:

label(*name=[name], selection=[string|image], font=[font]*)

name: indica el nombre de la etiqueta

selection: puede ser una cadena o una imagen que queremos que sea mostrada en el lugar donde aparece la etiqueta.

font: permite indicar la fuente en la que queremos mostrar el texto

```
new(L, label(nombre, 'texto que queramos que sea mostrado')),
```

WINDOW

Esta clase sirve para crear ventanas donde dibujar o poner otros objetos graficos

Constructor:

window(*label=[name], size=[size], display=[display]*)

name: indica el nombre de la ventana

size: indica el tamaño que queremos tenga la ventana

display: indica donde queremos que aparezca (recomendamos no usar este parametro)

Por ejemplo para crear una nueva ventana (de gráficos) de tamaño 320x200 pixels

```
new(W, window('nombre ventana', size(320, 200)),
```

Métodos:

Para esta clase nos interesarán básicamente los métodos 'display' y'flush':

display(figure, point): sirve para mostrar una figura en un lugar determinado de la ventana una determinada figura (que por ejemplo puede ser una imagen) y point que indicará las coordenadas (x,y) de la ventana donde queremos que se muestre.

El siguiente ejemplo mostraría la figura Figure en la posicion (22, 32) de la ventana W.

```
send(W, display, Figure, point(22,32))
```

flush(): sirve para redibujar la venana que invoca al método, es decir, si estamos realizando una serie de llamadas y no está en el bucle principal de la ventana, entonces podemos llamar a este método para redibujar la ventana y los elementos que hayamos creado o movido.

```
send(W, flush)
```

USANDO IMÁGENES: figure, bitmap, image, resource

Existen varias formas de mostrar imágenes en pantalla. XPCE/Prolog soporta varios formatos, y nos permite tanto cargar imágenes de disco, como guardar nuevas imágenes en el disco. Los formatos que permite para iconos, cursores y figuras son XPM, ICO y CUR; y los formatos para imágenes son JPEG, GIF, BMP y PNM.

En la web de XPCE se recomienda usar el formato XPM (X PixMap) debido a que se puede usar tanto para imágenes, como para iconos y cursores.

Bueno, ya centrados un poco, los pasos a seguir para cargar una imagen y mostrarla en una ventana previamente definida (en adelante W), deberemos, antes de nada, decirle donde van a estar nuestras imágenes, ya que por defecto busca en un directorio interno al programa.

Para indicarle un nuevo directorio donde tiene que buscar las imágenes (si no ha encontrado esas imágenes en todos los directorios que tuviera anteriormente), debemos usar la función **pce_image_directory**. Por ejemplo, para indicarle que queremos buscar en el

directorio actual donde hemos hecho el *consult*, deberemos indicarselo del siguiente modo (preferiblemente al principio del código, y fuera de cualquier regla o hecho):

```
:- pce_image_directory('./')
```

Una vez hecho esto ya sabe donde encontrar las imágenes, ahora deberemos indicarle que imágenes queremos que cargue. Esto podemos hacerlo con los **resources**. Básicamente su uso se resume en la siguiente linea:

resource(name, class, image): donde el nombre sirve para referenciar posteriormente al objeto cargado, la clase indica el tipo de objeto que vamos a tratar (en este caso siempre será ***image,*** y el tercer campo indica la imagen que queremos cargar (en este caso).

Por ejemplo si quisiéramos cargar la imagen "fondo.jpg" que se encuentra dentro del directorio "fondos" relativo a donde tenemos el archivo de prolog, entonces deberiamos escribir algo como:

```
:- pce_image_directory('./fondos').
resource(fondo, image, image('fondo.jpg')).
```

En este caso, le asignaríamos el nombre "fondo" para referenciarlo posteriormente.

Con el predicado **image** únicamente sirve para que cargue dicha imagen. Es decir, lo que hacemos es asociarle dicha imagen al nombre 'fondo' que despues usaremos como referencia cuando queramos dibujar esa imagen.

Ahora una vez tenemos definidos todos los resources, ya podemos proceder a mostrar una imagen en pantalla (en una ventana W que hayamos creado). Para ello necesitamos convertir esta imagen en un bitmap, y posteriormente insertar el bitmap en una figura, y esta figura ya la podremos mostrar.

Veamos esto con un ejemplo. Primero crearemos la figura, y el mapa de bits, usando el constructor del mapa de bits para decirle que imagen es la que queremos que convierta. Así pues haremos algo como:

```
new(Fig, figure),
new(Bitmap, bitmap(resource(fondo), @on)),
```

Al hacer *resource(fondo)*, él internamente sabe que nos referimos a la imagen definida anteriormente a la que le hemos asignado el nombre 'fondo', el parámetro **@on** indica que queremos que se conserven las transparencias de esa imagen, si pusiéramos

@off o si directamente omitiéramos este parámetro, entonces no se admitirían las transparencias.

Ahora ya tenemos el mapa de bits y la figura creadas y solo queda insertar la imagen en la figura, y mostrarla. Antes de mostrar la figura en la ventana deberemos poner el siguiente código:

```
send(Bitmap, name, 1),
send(Fig, display, Bitmap),
send(Fig, status, 1),
```

Estas líneas son imprescindibles, ya que sino no se verá la imagen. La segunda de las líneas sirve para que muestre el mapa de bits en la figura. La primera si no se pone no se ve la imagen. La tercera de las líneas sirve para que la Figura tenga el estado 1.

Así pues ahora solamente queda decirle que muestre la imagen en la ventana, lo cual se realiza mediante el método **display** de la clase **window** ya comentado anteriormente.

Tras explicar todo esto, se puede hacer un predicado que cargaría imágenes, y no tener que preocuparnos por crear figuras cada vez, así pues:

```
nueva_imagen(Ventana, Figura, Imagen, Posicion) :-
        new(Figura, figure),
        new(Bitmap, bitmap(resource(Imagen),@on)),
        send(Bitmap, name, 1),
        send(Figura, display, Bitmap),
        send(Figura, status, 1),
        Send(Ventana, display, Figura, Posicion).
```

Donde **Imagen** indica el nombre que le hemos dado al recurso, **Posición** es un punto del tipo de *point(23,34)*, **Figura** devolverá la figura que ha creado y **Ventana** es la ventana donde queremos que se dibuje. Por lo tanto, una posible llamada seria:

```
nueva_imagen(W, Figura, fondo, point(43, 225)),
```

Cargaria la imagen ‘fondo’ en la posicion (43, 225) de la ventana W, devolviendo la figura donde se ha creado.

Ahora únicamente haría falta indicar como mover imágenes que ya han sido mostradas en la ventana.

Esto es tan fácil como utilizar los metodos **move** y **relative_move**, explicados en la documentación. Sin embargo, a continuacion pongo dos predicados que sirven de interfaz, y servirían para mover la imagen a un punto concreto de la ventana (move) o desplazar esa figura mediante un vector de desplazamiento (relative_move).

```
mover_imagen_a(Figure, X, Y) :-
        send(Figure, move, point(X, Y)).

mover_imagen(Figure, X, Y) :-
        send(Figure, relative_move, point(X, Y)).
```

Donde **mover_imagen_a** mueve la figura al punto X, Y, y **mover_imagen** desplazaria la figura en las cantidades indicadas por X e Y.

Para más información:

?? ***Programming in XPCE/Prolog***: Guía de usuario para aprender a programar en *Prolog* con el XPCE, desde lo más básico hasta lo más complejo.
http://www.swi.psy.uva.nl/projects/xpce/UserGuide/

?? ***Class summary descriptions***: Página donde se puede encontrar información sobre gran numero de clases y algunos de sus métodos, con algún ejemplo.
http://www.swi.psy.uva.nl/projects/xpce/UserGuide/summary.html

?? ***Pagina principal de SWI-Prolog***: Página de donde descargar un interprete o la documentación para programar en Prolog y XPCE
http://www.swi-prolog.org/

?? ***The XPCE online reference manual:*** manual de referencia con todos los metodos y objetos que pueden ser creados y referenciados.
http://gollem.swi.psy.uva.nl:8080/