

Expansão Lógica

Programando a interface gráfica com:

XPCE / Prolog

Pau Sánchez Campello
4rto Ingenieria en Informatica.

1.- Carregando a biblioteca do PCE

Para usar predicados para trabalhar com gráficos no *Prolog*, teremos que carregar a biblioteca PCE. Para carregar esta biblioteca, vamos colocar a seguinte linha em qualquer parte do arquivo (de preferência no começo):

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

Esta linha está dizendo ao Prolog para carregar a biblioteca depois de terminar a compilação, antes que o prompt do Prolog nos permita fazer qualquer pergunta.

Uma vez carregada esta biblioteca, já temos uma série de predicados para criar janelas, botões, ... e uma variedade de objetos.

2.- Criando objetos e interagindo com eles

Com a biblioteca PCE trabalhamos com um esquema orientado a objetos, onde podemos criar classes e trabalhar com objetos diferentes, podendo chamar métodos desses objetos (passando-os pelos parâmetros correspondentes) ou chamar métodos que nos devolvam algum valor, e obviamente se criarmos objetos, também podemos destruí-los.

Portanto, existem principalmente 4 predicados como que você pode trabalhar com XPCE / Prolog. Esses predicados são usados para levantar objetos, enviar mensagens para objetos, receber mensagens de objetos e liberar memória de objetos. Esses 4 predicados:

?? *new(?Reference, +NewTerm)*: Esse predicado coleta dois parâmetros, o primeiro pegaria a referência que é atribuída ao novo objeto, uma vez que **new** é usado para criar objetos. O segundo parâmetro indicaria o objeto que você deseja criar.

?? *send(?Receiver, +Selector(...Args...))*: O primeiro parâmetro do predicado é uma referência ao objeto para o qual queremos enviar uma mensagem. O segundo parâmetro indicará o método para o qual queremos chamar, o qual indicará ao lado dos argumentos que queremos enviar ao método.

?? *get(?Receiver, +Selector(+Argument...), -Result)*: O primeiro parâmetro do predicado é uma referência ao objeto para o qual queremos enviar uma mensagem. O segundo parâmetro indicará o método para o qual queremos chamar, o qual indicará ao lado dos argumentos que queremos enviar ao método.

?? *free(?Reference)*: libera a memória associada ao objeto indicado no primeiro parâmetro.

As referências são usadas para saber a que objeto nos referimos, portanto, cada objeto que acreditamos deve ter sua própria referência, já que, depois de toda a memória que reservamos com **new**, será conveniente liberá-lo **free**.

O Prolog usa principalmente dois tipos de referências, uma que seria através das variáveis típicas do prolog (uma string que começa com uma letra maiúscula, como Variable, Pepe,), e a outra maneira é definir referências nomeadas, que uma vez definido não podemos criar outra referência para esse nome, uma vez que nos dará um erro na execução. Estas últimas referências são interessantes para acessar globalmente o mesmo objeto, sem ter que passar nenhum tipo de parâmetro. Essas variáveis são criadas usando

o operador especial **@** então, qualquer nome que comece com **@** será uma variável associada ao nome que damos (por exemplo, **@pepe**, **@variable**).

Quando usamos as variáveis nomeadas, devemos ter um cuidado especial, pois devemos liberá-las usando **free** antes do final da avaliação do predicado, porque se retornarmos à mesma consulta, e criarmos objetos usando essas mesmas variáveis, ela falhará e não irá parar executar.

Por exemplo, para criar uma caixa de diálogo que contenha um botão e ao clicar nela, feche essa janela:

```
ejemplo :-  
/*  
 * Crie o diálogo de objeto na variável D  
 */  
new(D, dialog('Nombre del  
Dialogo')),  
  
/* * Crie o objeto de botão armazenando a variável @boton de tal maneira  
 * 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))),  
  
/*  
 * Insira o botão na caixa de diálogo  
 */  
send(D, append(@boton)),  
  
/*  
 * Envie a mensagem open para o diálogo para criar e mostrar a janela.  
 */  
send(D,  
open).
```

Neste exemplo, podemos ver como os novos predicados *new*, *free* e *send* são usados, assim como podemos ver que outros aparecem como *append*, *open* e *quit*, que são métodos da classe *dialog* (dos quais criamos uma instância na variável D usando o predicado *new*).

Ao passar parâmetros na criação ou na chamada para um método de um objeto, esses métodos podem ter dois tipos de parâmetros, eles podem ter parâmetros obrigatórios e parâmetros opcionais. A diferença é que, se não indicarmos explicitamente um parâmetro obrigatório, o XPCE gerará um erro, enquanto, se não definirmos um parâmetro opcional, o XPCE colocará em vigor a constante **@default** cujo valor já é aquele definido por padrão nesse método.

3.- Envio de mensagens

Como vimos no exemplo, certos objetos podem executar certas ações, como o caso do objeto **button**, então quando clicamos no botão, podemos ter uma ou mais ações executadas, caso em que um ou mais predicados.

Se quisermos apenas avaliar um predicado, podemos usar o predicado:

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

Como você pode ver, o primeiro parâmetro indica quem é o receptor, nesse caso, no exemplo anterior, será sempre um objeto que criamos anteriormente com o novo. O seletor (segundo parâmetro) indicaria o nome do método ou função ou predicado que você deseja chamar, e os parâmetros subsequentes são os argumentos dessa função.

Como uma nota importante, deve-se notar que apenas objetos declarados no XPCE podem entrar nesses parâmetros, ou seja, não podemos colocar uma lista como terceiro ou quarto parâmetro e esperar que essa lista seja passada para a função quando for chamada (pelo menos em princípio).

No campo do receptor podemos basicamente especificar uma variável que encontramos inicializada com new, tendo então necessariamente que invocar um método daquele objeto, ou ao contrário, também é permitido invocar um predicado com seus parâmetros correspondentes (exceto listas e tipos não básico que implementa o XPCE), mas neste caso no campo receptor teremos que colocar **@prolog** e no segundo parâmetro o nome do predicado e continuar preenchendo os argumentos.

Para mostrar e entender melhor como funciona, você pode ver o seguinte exemplo:

```

ejemplo_mensajes :-  

    % Crie o objeto de diálogo na variável D  

    new(D, dialog('Nome del Dialogo')),  

  

    % Crie um botão que chame o predicado para mostrar a mensagem  

    new(B, button('Mostrar en Consola',  

  

        message(@prolog, mostrar_mensaje, 'Este es el valor que tendra la variable P'))),  

  

    % Crie um botão para fechar a caixa de diálogo  

    new(@boton, button('Cerrar Dialogo',  

  

        and(  

            message(D, destroy),  

            message(D, free),  

            message(D, free),  

            message(@boton,  

                free)))),  

  

    % insire os botões na caixa de diálogo  

    send(D, append(@boton)),  

    send(D, append(B)),  

  

    % Mostrar a janela  

    send(D, open).  

  

% Exibe uma mensagem no console.  

mostrar_mensaje(P) :-  

    write('La variable P vale '), write(P), nl.

```

4.- Criando elementos no ambiente gráfico

Uma vez que os conceitos básicos foram explicados, podemos criar objetos no ambiente gráfico, como diálogos, textos, rótulos, caixas de desenho, botões, etc ... Além disso, certamente nesta seção o uso de todos os itens acima será melhor compreendido.

Como dissemos antes, podemos criar vários tipos de objetos, então listarei alguns desses objetos e alguns de seus métodos.

OBJETO	DESCRIPCION
dialog	Usado para criar uma caixa de diálogo
button	Serve para criar um botão
cursor	É usado para modificar a imagem do cursor
figur	Ele é usado para colocar imagens na tela
imag	Serve para carregar imagens
bitma	Para converter uma imagem em um elemento gráfico (basicamente age como uma ponte para ir da imagem à figura)
pixmap	É praticamente equivalente a imagem
label	Para escrever um rótulo na tela (pode ser usado para mostrar um texto)
menu	Para criar um menu
menu_ba	Para criar uma barra de menu
menu_item	Para incorporar elementos em um menu
point	Para criar um ponto com 2 coordenadas
popu	Para criar um pop-up em uma janela.
slider	Para criar um displacer
window	Para criar uma janela onde você pode desenhar outra série de objetos, como imagens, etc ...

Das classes citadas, descreverei apenas o construtor e alguns métodos de alguns deles, e como usá-los ou interagir com as outras classes.

DIALOG

Esta é a classe básica para criar diálogos.

Constructor:

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

name: indica o título para a janela

size: é do tamanho do tipo e é usado para indicar o tamanho da janela

display: indica onde queremos que seja exibido (melhor não tocar neste parâmetro se não soubermos o que estamos fazendo)

Podemos ver que todos os parâmetros são opcionais, embora seja sempre bom dar um título à janela

Então, como exemplo, vamos criar um diálogo com o título 'Título do diálogo' e tamanho 440 x 320.

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

Métodos:

Esta classe tem vários métodos que podem ser interessantes, entre eles temos:

append(Objeto): Insira o objeto Objeto na caixa de diálogo, visualizando-o o mesmo, por exemplo, serve para inserir um botão ou outra coisa, como no seguinte exemplo:

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

open(): abre a caixa de diálogo exibindo-a na tela:

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

destroy(): fecha a janela de diálogo na tela:

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

BUTTON

Esta é a classe básica para criar botões

Constructor:

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

name: indica o nome do botão (se você não especificar o rótulo que deseja que o botão tenha, ele adotará um rótulo com o mesmo texto que o nome)

message: indica a mensagem ou ação que queremos executar quando clicamos no botão com o mouse.

label: Indique o rótulo que queremos que seja exibido no botão.

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

LABEL

Esta é a classe básica para criar etiquetas de texto

Constructor:

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

name: indica o nome da etiqueta

selection: Pode ser uma cadeia ou uma imagem que queremos mostrar no local onde a etiqueta aparece.

font: permite indicar a fonte na qual queremos mostrar o texto

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

WINDOW

Esta classe é usada para criar janelas para desenhar ou colocar outros objetos gráficos

Constructor:

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

name: indica o nome da janela

size: indica o tamanho que queremos que a janela tenha

display: indica onde queremos que apareça (recomendamos não usar este parâmetro)

Por exemplo, para criar uma nova janela (de gráficos) de tamanho 320x200 pixels

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

Métodos:

Para esta classe, estamos basicamente interessados nos métodos 'display' e 'flush':

display(figure, point): Vá para mostrar uma figura em um determinado lugar da janela uma determinada figura (que por exemplo pode ser uma imagem) e ponto que irá indicar as coordenadas (x, y) da janela onde queremos que ele seja exibido.

O exemplo a seguir mostra a figura Figure na posição (22, 32) da janela W.

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

flush(): Ele serve para redesenhar a janela que invoca o método, ou seja, se estamos fazendo uma série de chamadas e não está no loop da janela principal, então podemos chamar esse método para redesenhar a janela e os elementos que criamos ou movemos.

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

USANDO IMAGENS: figura, bitmap, imagem, recurso

Existem várias maneiras de exibir imagens na tela. O XPCE / Prolog suporta vários formatos e nos permite carregar imagens de disco, bem como salvar novas imagens em disco. Os formatos que permitem ícones, cursores e figuras são XPM, ICO e CUR; e os formatos para imagens são JPEG, GIF, BMP e PNM.

No site da XPCE, recomenda-se usar o formato XPM (X PixMap), pois ele pode ser usado para imagens, ícones e cursores.

Bem, já focado um pouco, os passos a seguir para carregar uma imagem e exibi-la em uma janela previamente definida (doravante W), devemos, antes de mais nada, dizer onde estarão nossas imagens, pois por padrão ela procura em um diretório interno ao programa.

Para indicar um novo diretório onde você deve procurar as imagens (se você não encontrou essas imagens em todos os diretórios que você tinha anteriormente), devemos usar a função **pce_image_directory**. Por exemplo, para indicar que queremos pesquisar no diretório atual em que fizemos a consulta, devemos indicá-lo da seguinte maneira (preferencialmente no início do código e fora de qualquer regra ou fato):

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

Feito isso, você sabe onde encontrar as imagens, agora devemos indicar as imagens que queremos carregar. Isso nós podemos fazer com os **resources**. Basicamente, seu uso é resumido na seguinte linha:

resource(name, class, image): onde o nome serve para referenciar posteriormente o objeto carregado, a classe indica o tipo de objeto a ser tratado (nesse caso sempre será *image*, e o terceiro campo indica a imagem que queremos carregar (neste caso).

Por exemplo, se quisermos carregar a imagem "fondos.jpg" que está dentro do diretório "fondos" em relação ao local em que temos o arquivo prolog, devemos escrever algo como:

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

Nesse caso, você atribui o nome "fondo" para referência posterior.

Com o predicado **image** só serve para carregar essa imagem. Ou seja, o que fazemos é associar essa imagem ao nome 'fondo' que usaremos mais tarde como referência quando quisermos desenhar essa imagem.

Agora, uma vez definidos todos os resources,, podemos mostrar uma imagem na tela (em uma janela W que criamos). Para isso, precisamos converter essa imagem em um bitmap e, em seguida, inserir o bitmap em uma figura, e essa figura podemos mostrá-lo.

Vamos ver isso com um exemplo. Primeiro, criaremos a figura e o bitmap, usando o construtor de bitmap para informar qual imagem queremos converter.

Então faremos algo como:

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

Ao fazer *resource(fondo)*, ele internamente sabe que nos referimos à imagem previamente definida à qual atribuímos o nome 'fondo', o parâmetro **@on** indica que queremos que as transparências dessa imagem sejam preservadas, se colocarmos

@off ou se omitirmos este parâmetro diretamente, as transparências não serão permitidas.

Agora temos o bitmap e a figura criada e só resta inserir a imagem na figura e mostrá-la. Antes de mostrar a figura na janela, teremos que colocar o seguinte código:

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

Estas linhas são essenciais, caso contrário a imagem não será vista. A segunda linha serve para mostrar o bitmap na figura. O primeiro, se não for colocado, a imagem não é visível. A terceira linha serve para fazer a figura ter o status 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).
```

Onde **Image** indica o nome que demos ao recurso, **Position** é um ponto do tipo *point(23,34)*, a **figura** retornará a figura que foi criada e **Window** é a janela onde queremos que seja desenhada. Portanto, uma possível chamada seria:

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

Eu carregaria a imagem 'fondo' na posição (43, 225) da janela W, retornando a figura onde ela foi criada.

Agora você só precisa indicar como mover imagens que já foram mostradas na janela.

Isso é tão fácil quanto usar os métodos **move** e **relative_move**, explicados na documentação. No entanto, eu coloco dois predicados que servem como uma interface e sirvo para mover a imagem para um ponto específico da janela (mover) ou movê-la usando um vetor de deslocamento (relative_move).

```
mover_imagen_a(Imagen, X, Y) :-  
    send(Imagen, move, point(X, Y)).  
  
mover_imagen(Imagen, X, Y) :-  
    send(Imagen, relative_move, point(X, Y)).
```

Onde **move_imagen_a** move a figura para o ponto X, Y e **move_image** moveria a figura nas quantidades indicadas por X e Y.

Para mais informações:

?? ***Programming in XPCE/Prolog:*** Guia do usuário para aprender a Programa em Prolog com o XPCE, do mais básico ao mais complexo. <http://www.swi.psy.uva.nl/projects/xpce/UserGuide/>

?? ***Class summary descriptions:*** Página onde você pode encontrar informações sobre um grande número de classes e alguns de seus métodos, com exemplos.
<http://www.swi.psy.uva.nl/projects/xpce/UserGuide/summary.html>

?? ***Pagina principal de SWI-Prolog:*** Página de onde baixar um intérprete ou a documentação para programar em Prolog e XPCE
<http://www.swi-prolog.org/>

?? ***The XPCE online reference manual:*** manual de referência com todos métodos e objetos que podem ser criados e referenciados.
<http://gollem.swi.psy.uva.nl:8080/>