# Taller de Programación Concurrente por Paso de Mensajes

**Modelos y Paradigmas de Programación**

*Harold Armando Achicanoy Estrella*

Noviembre 5 de 2017

## Punto 1

*Implementando puertos con celdas*

La implementación de puertos con celdas se realizó de la siguiente manera:

1. Definiendo el puerto con una celda `P` que estará asociada al nombre `S`. Con estos dos parámetros, se puede definir el procedimiento `NuevoPuerto` cuyo estado inicial es `S`, es decir, el contenido del puerto está asociado a la variable declarativa no ligada `S` que hace las veces de variable de flujo de datos al momento de su declaración.
2. Una vez el procedimiento `NuevoPuerto` crea el puerto de interés, se puede enviar información a través de él con el procedimiento `Enviar` que asocia el contenido inicial de la celda `P` utilizando la variable `Ant` para hacer la concatenación del nuevo envío de información dado por el contenido de la variable `X` y la variable declarativa `Z` sin ligar en su cola, para así construir la variable de flujo de datos `S` asociada al puerto `P`.

```
declare
fun {NuevoPuerto S}
   P = {NewCell S} % P: celda que simula el puerto
   proc {Enviar X} % {Enviar P X}
      % Z: variable sin ligar
      % Ant: se liga con el contenido antiguo de la celda P
      % para hacer la concatenacion de X y Z
      Z Ant in {Exchange P Ant Z}
      Ant=X|Z
   end
in
   proc {$ Msj} % Invocacion del procedimiento Enviar
      case Msj
      of enviar(X) then
         {Enviar X}
      end
   end
end
```

Una vez se cuenta con la creación del puerto como TAD con estado, empaquetado y seguro (con procedimiento de despacho), se crea la desempaquetada de la función `Enviar` como se muestra a continuación:

```
declare
proc {Enviar P X}
   {P enviar(X)}
end
```

Y a continuación se presentan los resultados obtenidos:

```
%% Prueba
declare S P
P = {NuevoPuerto S}
{Browse S}
{Enviar P a}
{Enviar P b}
{Enviar P c}
{Enviar P a}
```

```
a|b|c|a|_
```

**Punto 2**

*Implementando celdas con puertos*

**Punto 3**

*Contador con múltiples clientes*

La creación del contador a partir de un proceso cliente-servidor se desarrolló de la siguiente manera:

1. Definición de la función Contador, cuya variable de entrada corresponde a un listado de caracteres a partir del cual se contará la frecuencia de cada carácter y retornará el listado de caracteres únicos con su respectivo conteo.

```
declare
fun {Contador Xs}
   local
      fun {ListarContar Patron Xs}
         case Xs of X|Xr then
            case X of A#B then
               if A==Patron then A#B+1|Xr
               else A#B|{ListarContar Patron Xr}
               end
            else nil end
         else Patron#1|nil end
      end
      fun {Acumulador Xs Patron} Tupla in
         case Xs of X|Xr then
            Tupla={ListarContar X Patron}
            Tupla|{Acumulador Xr Tupla}
         else nil end
      end
   in
      thread {Acumulador Xs nil} end
   end
end
```

2. Una vez definida la función contador se procedió a la creación de un procedimiento que mezcla la generación del servidor a partir de un puerto `Servidor` sobre el cual 4 clientes enviaran las respectivas cadenas de caracteres de manera asincrónica (a través de 4 hilos) utilizando la el procedimiento `Forward` que envía los mensajes recibidos de cada cliente al puerto `Servidor`.

```
declare
proc {Merge L1 L2 L3 L4 Sen}
   Servidor={NewPort Sen} % Creacion del puerto Servidor
   {Browse {Contador Sen}}
   proc {Forward X}
      {Send Servidor X} % Envio de mensajes a traves del Servidor
   end
in % Envio de mensajes por 4 Clientes
   thread {ForAll L1 proc {$ X} {Forward X} {Delay 1000} end} end
   thread {ForAll L2 proc {$ X} {Forward X} {Delay 1000} end} end
   thread {ForAll L3 proc {$ X} {Forward X} {Delay 1000} end} end
   thread {ForAll L4 proc {$ X} {Forward X} {Delay 1000} end} end
end
```

Una vez definidas estos dos procedimientos se procede a probar el funcionamiento del programa a partir de los siguientes parámetros:

```
declare
L1=a|c|_
L2=b|c|_
L3=a|_
L4=a|a|b|c|_

{Browse {Merge L1 L2 L3 L4}}
```

Resultados correspondientes a dos ejecuciones diferentes

```
[a#1]|[a#1 b#1]|[a#2 b#1]|[a#3 b#1]|[a#4 b#1]|[a#4 b#1 c#1]|[a#4 b#1 c#2]|[a#4 b#2 c#2]|[a#4 b#2 c#3]|_
a|b|a|a|a|c|c|b|c|_<future>
[a#1]|[a#1 b#1]|[a#2 b#1]|[a#3 b#1]|[a#3 b#1 c#1]|[a#4 b#1 c#1]|[a#4 b#1 c#2]|[a#4 b#2 c#2]|[a#4 b#2 c#3]|_
a|b|a|a|c|a|c|b|c|_<future>
```

Como se observa, ejecuciones diferentes de los clientes generan diferentes resultados, esto debido al envío asincrónico de mensajes desde los hilos al puerto `Servidor`.

**Punto 4**

*Implementación de la función* `Portero`

La función `Portero` se implementó a partir de la abstracción `NuevoObjetoPuerto` adicionando las respectivas operaciones que actualizan el estado a retornar.

```
declare
fun {NuevoObjetoPuerto Inic}
   proc {MsgLoop S1 Estado}
      case S1 of Msg|S2 then
         case Msg % Adicion de funciones para generar el conteo
         of getIn(N) then {MsgLoop S2 Estado+N}
         [] getOut(N) then {MsgLoop S2 Estado-N}
         [] getCount(N) then N=Estado {MsgLoop S2 Estado}
         end
      [] nil then skip
      end
   end
   Sin
in
   thread {MsgLoop Sin Inic} end
   {NewPort Sin}
end
```

Prueba

```
%% Prueba
declare
Portero={NuevoObjetoPuerto 0} X Y Z
{Browse X#Y#Z}
{Send Portero getIn(5)}
{Send Portero getOut(2)}
{Send Portero getCount(X)}
{Send Portero getIn(2)}
{Send Portero getOut(5)}
{Send Portero getCount(Y)}
{Send Portero getIn(16)}
{Send Portero getOut(4)}
{Send Portero getCount(Z)}
```

```
3#0#12
```

*Implementación de la función* `Contador2`

Para esta implementación se separan las funciones al interior de la función `Contador`, con el fin de aplicar la abstracción `NuevoObjetoPuerto` como se ilustra a continuación:

```
declare
fun {ListarContar Patron Xs}
   case Xs of X|Xr then
      case X of A#B then
         if A==Patron then A#B+1|Xr
         else A#B|{ListarContar Patron Xr}
         end
      else nil end
   else Patron#1|nil end
end

declare
fun {NuevoObjetoPuerto Comp Inic}
   proc {MsgLoop S1 Estado}
      case S1 of Msg|S2 then
         {Browse Estado}
         {MsgLoop S2 {Comp Msg Estado}}
      [] nil then skip
      end
   end
   Sin
in
   thread {MsgLoop Sin Inic} end
   {NewPort Sin}
end
```

Prueba

```
declare S P
PuertoSalida={NuevoObjetoPuerto S P}
Contador2={NuevoObjetoPuerto
           fun {$ Patron Xs} Tupla in
              Tupla={ListarContar Patron Xs}
              {Send PuertoSalida Tupla}
              Tupla
           end nil}

%% Prueba
{Send Contador2 a}
{Send Contador2 a}
{Send Contador2 b}
{Send Contador2 c}
{Send Contador2 d}
```

```
[a#1]
[a#2]
[a#2 b#1]
[a#2 b#1 c#1]
```