Paradigmas Avanzados de Programación

El modelo de concurrencia por paso de mensajes

Juan Francisco Díaz Frias

Maestría en Ingeniería, Énfasis en Ingeniería de Sistemas y Computación
Escuela de Ingeniería de Sistemas y Computación,
home page: http://eisc.univalle.edu.co
Universidad del Valle - Cali, Colombia

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

1 El modelo
- Introducción
- Sintaxis
- El concepto de puerto
- Semántica de los puertos

2 Objetos puerto
- Objetos flujo
- Objetos puerto

Plan

1 El modelo
- Introducción
- Sintaxis
- El concepto de puerto
- Semántica de los puertos

2 Objetos puerto
- Objetos flujo
- Objetos puerto

Plan

1 El modelo
- Introducción
- Sintaxis
- El concepto de puerto
- Semántica de los puertos

2 Objetos puerto
- Objetos flujo
- Objetos puerto

Concurrencia por paso de mensajes

¿Qué es?

El paso de mensajes es un estilo de programación en el cual un programa consiste de entidades independientes que interactúan enviándose mensajes entre ellas, de manera asincrónica.

Áreas de importancia

- Sistemas multiagentes
- Sistemas distribuidos
- Construcción de sistemas confiables

Concurrencia por paso de mensajes

¿Qué es?

El paso de mensajes es un estilo de programación en el cual un programa consiste de entidades independientes que interactúan enviándose mensajes entre ellas, de manera asincrónica.

Áreas de importancia

- Sistemas multiagentes
- Sistemas distribuidos
- Construcción de sistemas confiables

Plan

1 El modelo
- Introducción
- Sintaxis
- El concepto de puerto
- Semántica de los puertos

2 Objetos puerto
- Objetos flujo
- Objetos puerto

El modelo concurrente por paso de mensajes

⟨d⟩ ::=	
skip	Declaración vacía
⟨d⟩$_1$ ⟨d⟩$_2$	Declaración de secuencia
local ⟨x⟩ **in** ⟨d⟩ **end**	Creación de variable
⟨x⟩$_1$=⟨x⟩$_2$	ligadura variable-variable
⟨x⟩=⟨v⟩	Creación de valor
if ⟨x⟩ **then** ⟨d⟩$_1$ **else** ⟨d⟩$_2$ **end**	Condicional
case ⟨x⟩ **of** ⟨patrón⟩ **then** ⟨d⟩$_1$ **else** ⟨d⟩$_2$ **end**	Reconocimiento de patrones
{ ⟨x⟩ ⟨y⟩$_1$ ··· ⟨y⟩$_n$ }	Invocación de procedimiento
thread ⟨d⟩ **end**	Creación de hilo
{NewName ⟨x⟩}	Creación de nombre
{NewPort ⟨y⟩ ⟨x⟩}	**Creación de puerto**
{Send ⟨x⟩ ⟨y⟩}	**Remisión al puerto**

Plan

1 El modelo
- Introducción
- Sintaxis
- El concepto de puerto
- Semántica de los puertos

2 Objetos puerto
- Objetos flujo
- Objetos puerto

El concepto de puerto

¿Qué es?

Un puerto es un TAD con dos operaciones, a saber, la creación de un canal y el envío de algo por él:

- `{NewPort ?S ?P}`: crea un puerto nuevo con punto de entrada `P` y flujo `S`.
- `{Send P X}`: concatena `X` al flujo correspondiente al punto de entrada `P`.

Ejemplo

```
declare S P in
{NewPort S P}
{Browse S}
{Send P a}
{Send P b}
```

FIFO

Las remisiones consecutivas desde el mismo hilo aparacen en el flujo en el mismo orden en que fueron ejecutadas.

El concepto de puerto

¿Qué es?

Un puerto es un TAD con dos operaciones, a saber, la creación de un canal y el envío de algo por él:

- `{NewPort ?S ?P}`: crea un puerto nuevo con punto de entrada `P` y flujo `S`.
- `{Send P X}`: concatena `X` al flujo correspondiente al punto de entrada `P`.

Ejemplo

```
declare S P in
{NewPort S P}
{Browse S}
{Send P a}
{Send P b}
```

FIFO

Las remisiones consecutivas desde el mismo hilo aparacen en el flujo en el mismo orden en que fueron ejecutadas.

El concepto de puerto

¿Qué es?

Un puerto es un TAD con dos operaciones, a saber, la creación de un canal y el envío de algo por él:

- `{NewPort ?S ?P}`: crea un puerto nuevo con punto de entrada `P` y flujo `S`.
- `{Send P X}`: concatena `X` al flujo correspondiente al punto de entrada `P`.

Ejemplo

```
declare S P in
{NewPort S P}
{Browse S}
{Send P a}
{Send P b}
```

FIFO

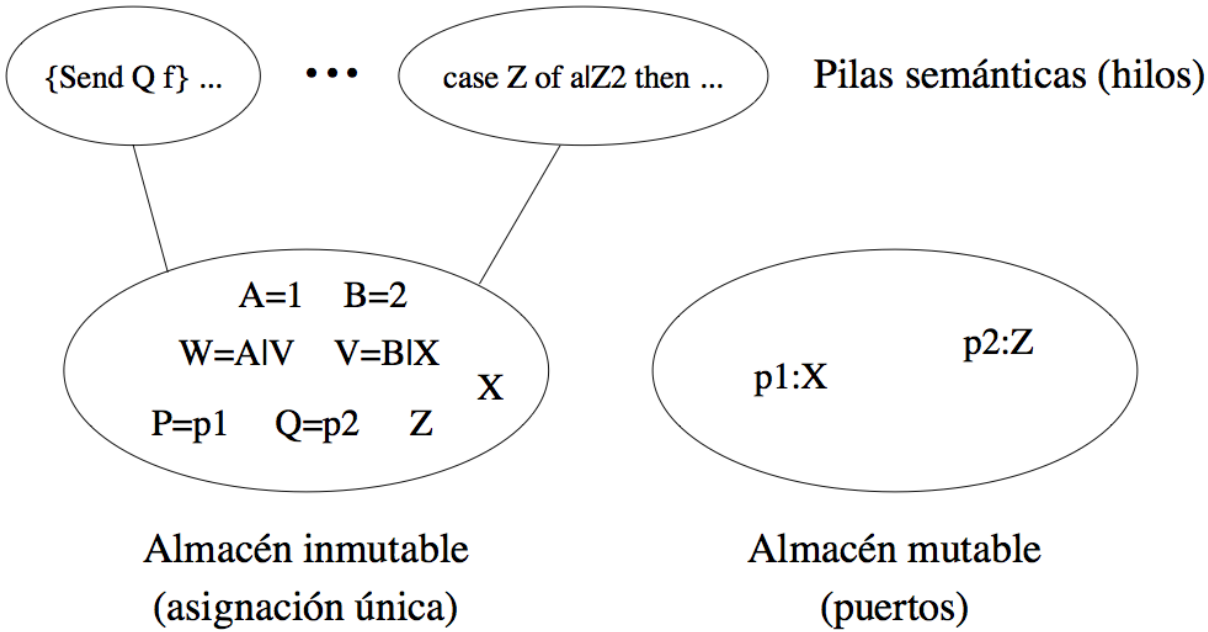
Las remisiones consecutivas desde el mismo hilo aparacen en el flujo en el mismo orden en que fueron ejecutadas.

Plan

1 El modelo
- Introducción
- Sintaxis
- El concepto de puerto
- Semántica de los puertos

2 Objetos puerto
- Objetos flujo
- Objetos puerto

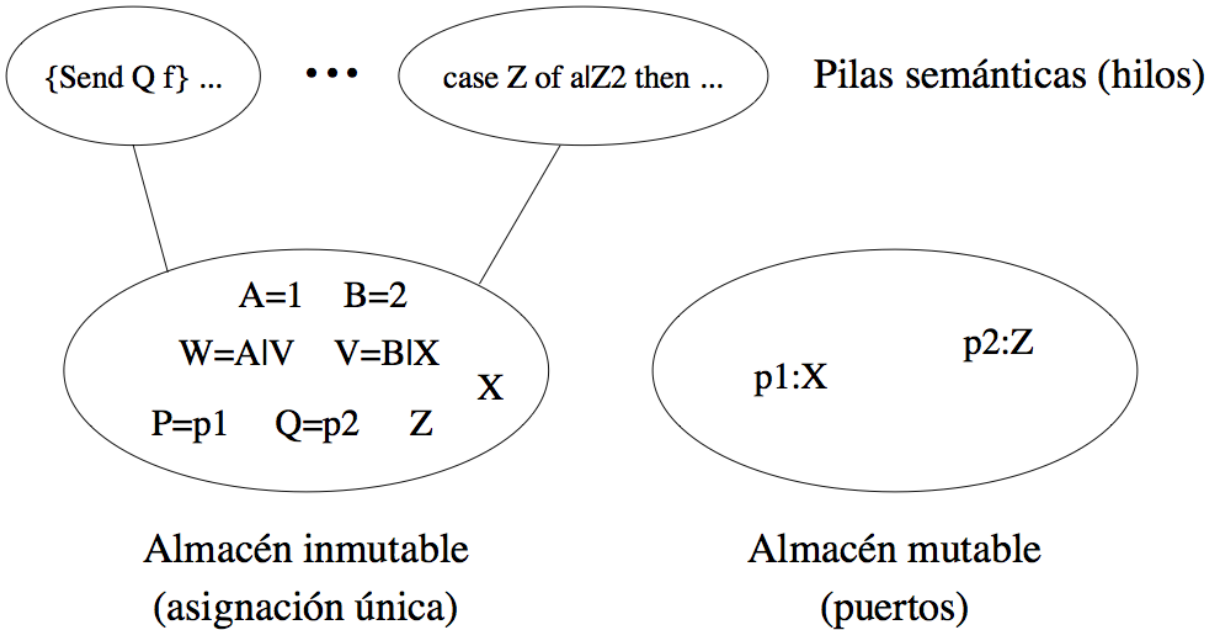
Semántica de los puertos



Estado

- Almacén nuevo, μ, denominado el almacén mutable.
- Un puerto es $x : y$, donde x y y son variables del almacén de asignación única.
- x siempre estará ligada a un valor de tipo nombre que representa un puerto y y siempre será no-ligada.
- El estado es una tripleta (MST, σ, μ)

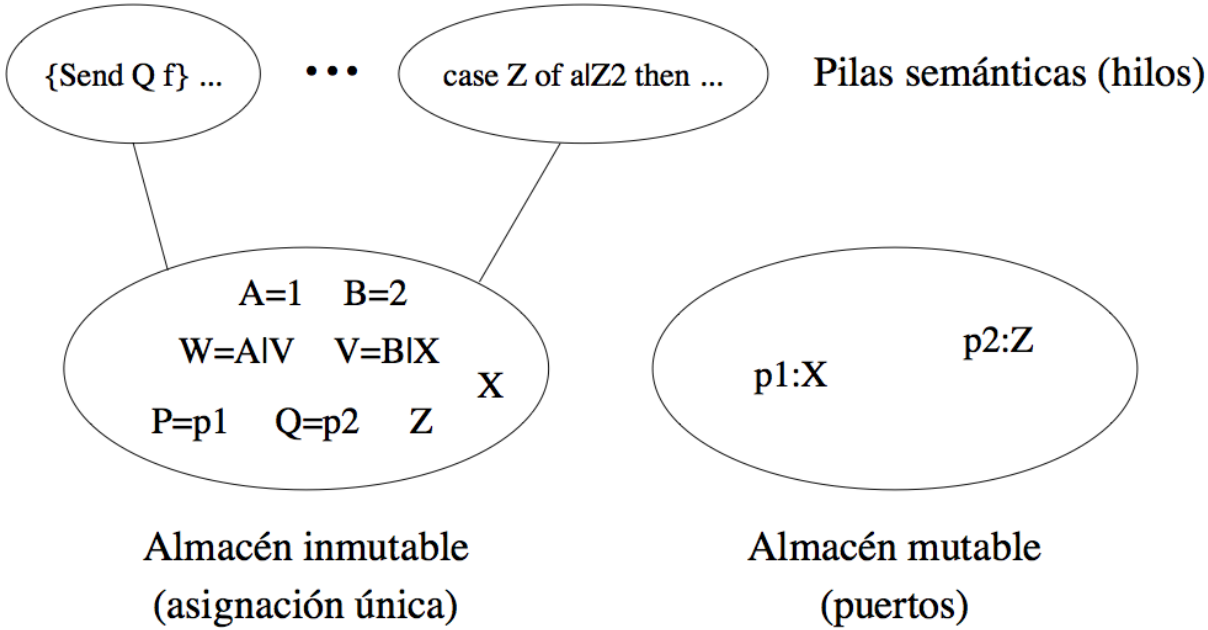
Semántica de los puertos



Estado

- Almacén nuevo, μ, denominado el almacén mutable.
- Un puerto es $x : y$, donde x y y son variables del almacén de asignación única.
- x siempre estará ligada a un valor de tipo nombre que representa un puerto y y siempre será no-ligada.
- El estado es una tripleta (MST, σ, μ)

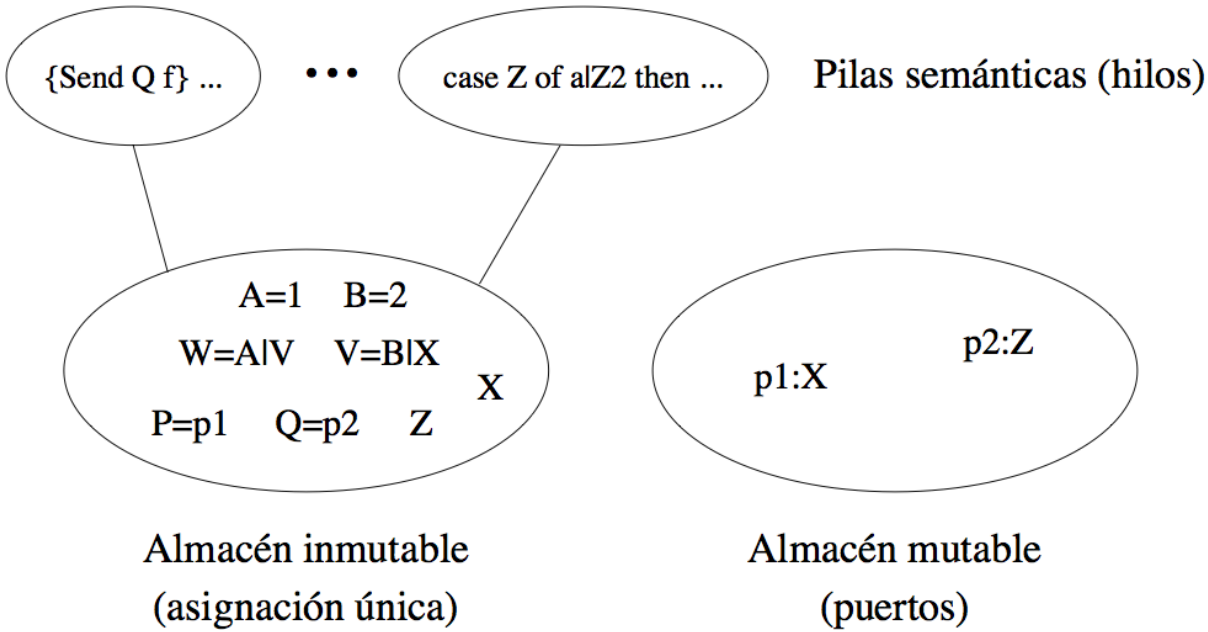
Semántica de los puertos



Estado

- Almacén nuevo, μ, denominado el almacén mutable.
- Un puerto es $x : y$, donde x y y son variables del almacén de asignación única.
- x siempre estará ligada a un valor de tipo nombre que representa un puerto y y siempre será no-ligada.
- El estado es una tripleta (MST, σ, μ)

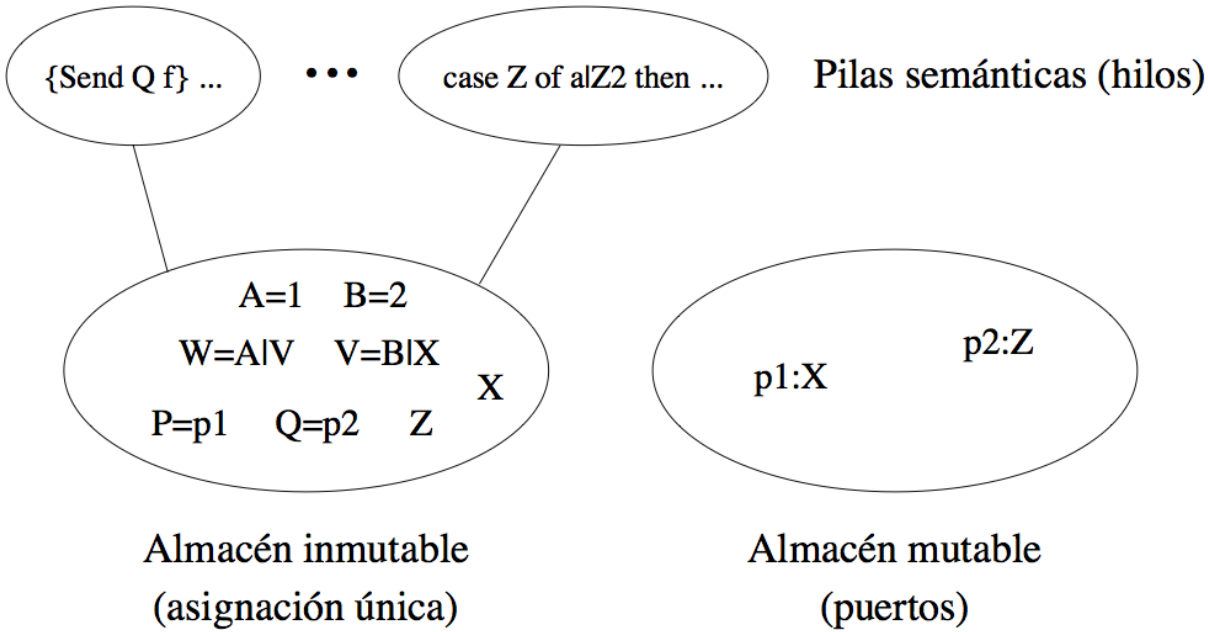
Semántica de los puertos



Estado

- Almacén nuevo, μ, denominado el almacén mutable.
- Un puerto es $x : y$, donde x y y son variables del almacén de asignación única.
- x siempre estará ligada a un valor de tipo nombre que representa un puerto y y siempre será no-ligada.
- El estado es una tripleta (MST, σ, μ)

Semántica de los puertos



Estado

- Almacén nuevo, μ, denominado el almacén mutable.
- Un puerto es $x : y$, donde x y y son variables del almacén de asignación única.
- x siempre estará ligada a un valor de tipo nombre que representa un puerto y y siempre será no-ligada.
- El estado es una tripleta (MST, σ, μ)

Semántica de `NewPort`

La operación `NewPort`

La declaración semántica $(\texttt{\{NewPort}\ \langle x \rangle\ \langle y \rangle\texttt{\}}, E)$ hace lo siguiente:

- Crea un nombre fresco para el puerto, n.
- Liga $E(\langle y \rangle)$ y n en el almacén.
- Si la ligadura es exitosa, entonces agrega la pareja $E(\langle y \rangle) : E(\langle x \rangle)$ al almacén mutable μ.
- Si la ligadura falla, entonces lanza una condición de error.

Semántica de `Send`

La operación `Send`

La declaración semántica $(\{\texttt{Send}\ \langle x\rangle\ \langle y\rangle\},E)$ hace lo siguiente:

- Si la condición de activación es cierta ($E(\langle x\rangle)$ está determinada), entonces se realizan las acciones siguientes:
  - Si $E(\langle x\rangle)$ no está ligada al nombre de un puerto, entonces se lanza una condición de error.
  - Si el almacén mutable contiene $E(\langle x\rangle) : z$, entonces se realizan las acciones siguientes:
    - Crear una variable nueva, z', en el almacén.
    - Actualizar el almacén mutable de manera que en lugar de la pareja $E(\langle x\rangle) : z$, aparezca la pareja $E(\langle x\rangle) : z'$.
    - Ligar z en el almacén con la lista nueva $E(\langle y\rangle) \mid z'$.
- Si la condición de activación es falsa, entonces se suspende la ejecución.

Plan

1 El modelo
- Introducción
- Sintaxis
- El concepto de puerto
- Semántica de los puertos

2 Objetos puerto
- Objetos flujo
- Objetos puerto

Recorderis: Objetos flujo

Definición

Un objeto flujo es un procedimiento recursivo que se ejecuta en un hilo propio y se comunica con otros objetos flujo a través de flujos de entrada y de salida.

¿Porqué objeto?

Porque es una sola entidad que combina las nociones de valor y operación. Esto en contraste con un TAD, en el cual los valores y las operaciones son entidades separadas

Abstracción

```
proc {ObjetoFlujo S1 X1 ?T1}
   case S1
   of M|S2 then N X2 T2 in
      {EstadoProximo M X1 N X2}
      T1=N|T2
      {ObjetoFlujo S2 X2 T2}
   [] nil then T1=nil end
end
declare S0 X0 T0 in
thread
   {ObjetoFlujo S0 X0 T0}
end
```

Recorderis: Objetos flujo

Definición

Un objeto flujo es un procedimiento recursivo que se ejecuta en un hilo propio y se comunica con otros objetos flujo a través de flujos de entrada y de salida.

¿Porqué objeto?

Porque es una sola entidad que combina las nociones de valor y operación. Esto en contraste con un TAD, en el cual los valores y las operaciones son entidades separadas

Abstracción

```
proc {ObjetoFlujo S1 X1 ?T1}
   case S1
   of M|S2 then N X2 T2 in
      {EstadoProximo M X1 N X2}
      T1=N|T2
      {ObjetoFlujo S2 X2 T2}
   [] nil then T1=nil end
end
declare S0 X0 T0 in
thread
   {ObjetoFlujo S0 X0 T0}
end
```

Recorderis: Objetos flujo

Definición

Un objeto flujo es un procedimiento recursivo que se ejecuta en un hilo propio y se comunica con otros objetos flujo a través de flujos de entrada y de salida.

¿Porqué objeto?

Porque es una sola entidad que combina las nociones de valor y operación. Esto en contraste con un TAD, en el cual los valores y las operaciones son entidades separadas

Abstracción

```
proc {ObjetoFlujo S1 X1 ?T1}
   case S1
   of M|S2 then N X2 T2 in
      {EstadoProximo M X1 N X2}
      T1=N|T2
      {ObjetoFlujo S2 X2 T2}
   [] nil then T1=nil end
end
declare S0 X0 T0 in
thread
   {ObjetoFlujo S0 X0 T0}
end
```

La plantilla `ObjetoFlujo`

`EstadoProximo`

Toma un mensaje de entrada `M` y un estado `X1`, y calcula un mensaje de salida `N` y un estado nuevo `X2`.

Ejecución

- Al ejecutar `ObjetoFlujo` en un hilo nuevo se crea un objeto flujo nuevo con flujo de entrada `S0`, flujo de salida `T0`, y estado inicial `X0`.
- El objeto flujo lee mensajes del flujo de entrada, realiza cálculos internos, y envía mensajes por el flujo de salida. En general, un objeto puede tener cualquier número fijo de flujos de entrada y de salida.

La plantilla `ObjetoFlujo`

`EstadoProximo`
Toma un mensaje de entrada `M` y un estado `X1`, y calcula un mensaje de salida `N` y un estado nuevo `X2`.

Ejecución

- Al ejecutar `ObjetoFlujo` en un hilo nuevo se crea un objeto flujo nuevo con flujo de entrada `S0`, flujo de salida `T0`, y estado inicial `X0`.
- El objeto flujo lee mensajes del flujo de entrada, realiza cálculos internos, y envía mensajes por el flujo de salida. En general, un objeto puede tener cualquier número fijo de flujos de entrada y de salida.

Ejemplo de uso

Enlace de objetos flujo

El siguiente es un canal compuesto por tres objetos flujo:

```
declare S0 T0 U0 V0 in
thread {ObjetoFlujo S0 0 T0} end
thread {ObjetoFlujo T0 0 U0} end
thread {ObjetoFlujo U0 0 V0} end
```

El primer objeto recibe de `S0` y envía por `T0`, el cual es recibido por el segundo objeto, y así sucesivamente.

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

1 El modelo
- Introducción
- Sintaxis
- El concepto de puerto
- Semántica de los puertos

2 Objetos puerto
- Objetos flujo
- Objetos puerto

Objetos puerto

Definición

Un objeto puerto es una combinación de uno o más puertos y un objeto flujo. Esto extiende los objetos flujo en dos formas.

Comunicación muchos a uno

Muchos hilos pueden referenciar un objeto puerto dado y enviarle mensajes a él de manera independiente. Esto no es posible con un objeto flujo porque éste tiene que conocer de dónde viene el próximo mensaje.

Los puertos pueden ser embebidos...

Los objetos puerto se pueden embeber dentro de estructuras de datos (incluyendo los mensajes). Esto no es posible con un objeto flujo pues éste está referenciado por un flujo que sólo puede ser extendido por un único hilo.

Objetos puerto

Definición

Un objeto puerto es una combinación de uno o más puertos y un objeto flujo. Esto extiende los objetos flujo en dos formas.

Comunicación muchos a uno

Muchos hilos pueden referenciar un objeto puerto dado y enviarle mensajes a él de manera independiente. Esto no es posible con un objeto flujo porque éste tiene que conocer de dónde viene el próximo mensaje.

Los puertos pueden ser embebidos...

Los objetos puerto se pueden embeber dentro de estructuras de datos (incluyendo los mensajes). Esto no es posible con un objeto flujo pues éste está referenciado por un flujo que sólo puede ser extendido por un único hilo.

Objetos puerto

Definición

Un objeto puerto es una combinación de uno o más puertos y un objeto flujo. Esto extiende los objetos flujo en dos formas.

Comunicación muchos a uno

Muchos hilos pueden referenciar un objeto puerto dado y enviarle mensajes a él de manera independiente. Esto no es posible con un objeto flujo porque éste tiene que conocer de dónde viene el próximo mensaje.

Los puertos pueden ser embebidos...

Los objetos puerto se pueden embeber dentro de estructuras de datos (incluyendo los mensajes). Esto no es posible con un objeto flujo pues éste está referenciado por un flujo que sólo puede ser extendido por un único hilo.

Objetos puerto

Puertos y Agentes

Algunas veces se usa la palabra “agente” para cubrir una idea similar: una entidad activa con la cual uno puede intercambiar mensajes.

¿Qué es un programa?

Un programa consiste de un conjunto de objetos puerto que envían y reciben mensajes.
Los objetos puerto pueden crear nuevos objetos puerto. También pueden enviar mensajes que contengan referencias a otros objetos puerto.
Esto significa que el conjunto de objetos puerto forma un grafo que puede evolucionar durante la ejecución.

Objetos puerto

Puertos y Agentes

Algunas veces se usa la palabra “agente” para cubrir una idea similar: una entidad activa con la cual uno puede intercambiar mensajes.

¿Qué es un programa?

Un programa consiste de un conjunto de objetos puerto que envían y reciben mensajes.
Los objetos puerto pueden crear nuevos objetos puerto. También pueden enviar mensajes que contengan referencias a otros objetos puerto.
Esto significa que el conjunto de objetos puerto forma un grafo que puede evolucionar durante la ejecución.

Estructura objetos puerto

Estructura

```
declare P1 P2 ... Pn in
local S1 S2 ... Sn in
   {NewPort S1 P1}
   {NewPort S2 P2}
   ...
   {NewPort Sn Pn}
   thread {PR S1 S2 ... Sn} end
end
```

Interpretación

El hilo contiene un procedimiento recursivo `PR` que lee los flujos del puerto y realiza ciertas acciones por cada mensaje recibido. Enviar un mensaje a un objeto puerto es simplemente enviar un mensaje a uno de sus puertos.

Primer ejemplo

Ejemplo

```
declare P in
local S in
   {NewPort S P}
   thread {ForAll S Browse} end
end
```

Invocación

Al invocar `{Send P hola}` se desplegará, finalmente, `hola`.

¿Cuál es la diferencia con los objetos flujo?

Primer ejemplo

Ejemplo

```
declare P in
local S in
   {NewPort S P}
   thread {ForAll S Browse} end
end
```

Invocación

Al invocar `{Send P hola}` se desplegará, finalmente, `hola`.

¿Cuál es la diferencia con los objetos flujo?

La abstracción `NuevoObjetoPuerto`

Objeto con un solo puerto

```
fun {NuevoObjetoPuerto Inic Fun}
Sen Ssal in
   thread {FoldL Sen Fun Inic Ssal} end
   {NewPort Sen}
end
```

Para definir el objeto puerto, sólo tenemos que pasar el estado inicial `Inic` y la función de transición de estados `Fun`.

La abstracción `NuevoObjetoPuerto`

Objetos reactivos

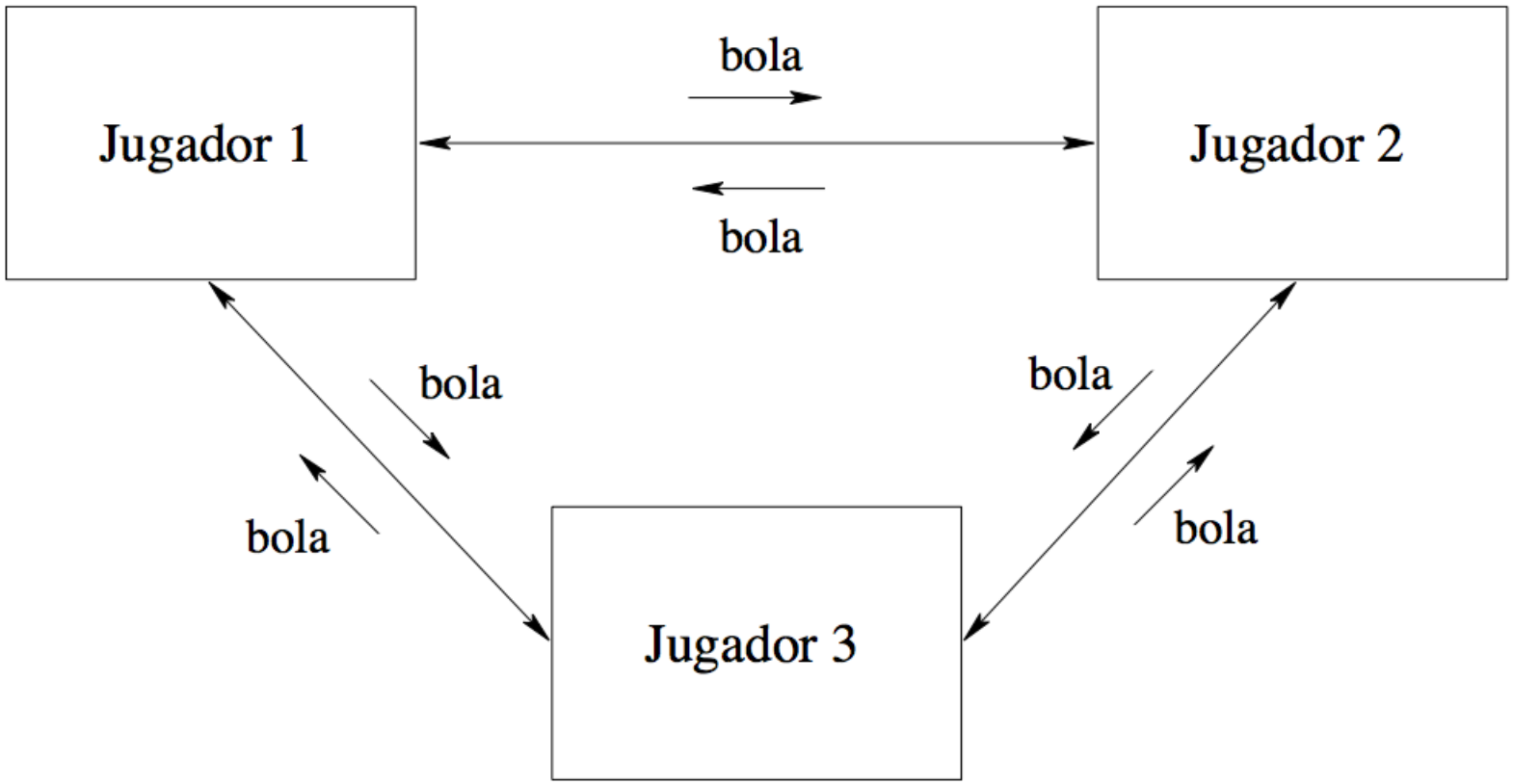
La abstracción es aún más sencilla para ellos, pues no se necesita ningún acumulador:

```
fun {NuevoObjetoPuerto2 Proc}
Sen in
   thread for Msj in Sen do {Proc Msj} end end
   {NewPort Sen}
end
```

No hay función de transición de estados, sino simplemente un procedimiento que es invocado con cada mensaje.

Ejemplo

Tres jugadores en círculo



Código

Creación de jugador nuevo

```
fun {Jugador Otros}
 {NuevoObjetoPuerto2
   proc {$ Msg}
    case Msg of bola then
      Ran={OS.rand} mod
          {Width Otros} + 1
     in
      {Send Otros.Ran bola}
     end
    end}
end
```

Configurar el juego

```
J1={Jugador otros(J2 J3)}
J2={Jugador otros(J1 J3)}
J3={Jugador otros(J1 J2)}
```

Iniciar el juego

```
{Send J1 bola}
```

Código

Creación de jugador nuevo

```
fun {Jugador Otros}
 {NuevoObjetoPuerto2
   proc {$ Msg}
    case Msg of bola then
      Ran={OS.rand} mod
          {Width Otros} + 1
     in
      {Send Otros.Ran bola}
     end
    end}
end
```

Configurar el juego

```
J1={Jugador otros(J2 J3)}
J2={Jugador otros(J1 J3)}
J3={Jugador otros(J1 J2)}
```

Iniciar el juego

```
{Send J1 bola}
```

Código

Creación de jugador nuevo

```
fun {Jugador Otros}
 {NuevoObjetoPuerto2
   proc {$ Msg}
    case Msg of bola then
      Ran={OS.rand} mod
          {Width Otros} + 1
     in
      {Send Otros.Ran bola}
     end
    end}
end
```

Configurar el juego

```
J1={Jugador otros(J2 J3)}
J2={Jugador otros(J1 J3)}
J3={Jugador otros(J1 J2)}
```

Iniciar el juego

```
{Send J1 bola}
```