Lección 14: Programación dirigida por eventos

- Introducción
- Arquitectura basada en eventos
  - Bus de eventos
- Una primera reflexión
- Programación basada en eventos
- Patrones de diseño basados en la idea de eventos
  - Aplicación de los patrones

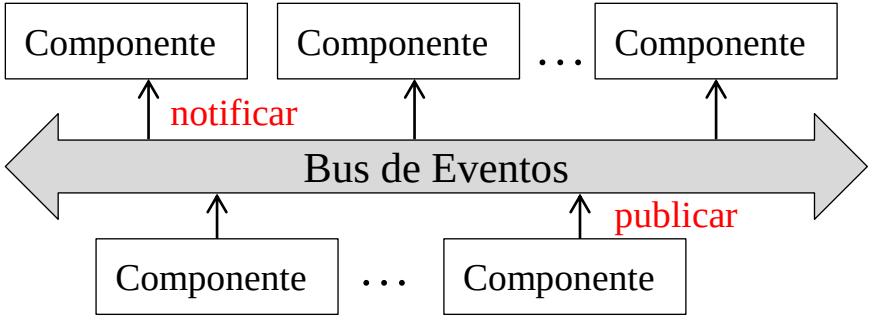
Introducción

- Un *sistema distribuido* consta de:
  - *Componentes*: entidades funcionales (interfaz bien definida)
  - *Conectores*: comunicación y coordinación entre componentes

- Una *arquitectura software* describe a nivel de diseño cómo construir un sistema distribuido
  - Existe una amplia gama de arquitecturas
  - Un elemento diferenciador clave entre las diferentes propuestas es el tipo de conector utilizado
    - Relación con los estilos de comunicación presentados en clase

Arquitectura basada en eventos

- Idea intuitiva:



- Comunicación basada en la propagación de eventos
- Procesos publicadores/suscriptores
- El bus gestiona la comunicación y garantiza la notificación de eventos a los procesos interesados

Arquitectura basada en eventos

- *Comunicación desacoplada** entre los componentes

- Publicadores de eventos:
  - No necesitan especificar a quién va dirigido en evento
  - No necesitan conocer cuántos receptores están interesados
  - Comunican el evento y continúan
- Suscriptores de eventos:
  - No necesitan saber quién publicó el eventos
  - No necesitan saber cuántos publicadores hay
  - Esperan una nueva notificación y reaccionan localmente

* *acoplamiento*: grado de dependencia de los componentes de un sistema entre sí

Bus de eventos

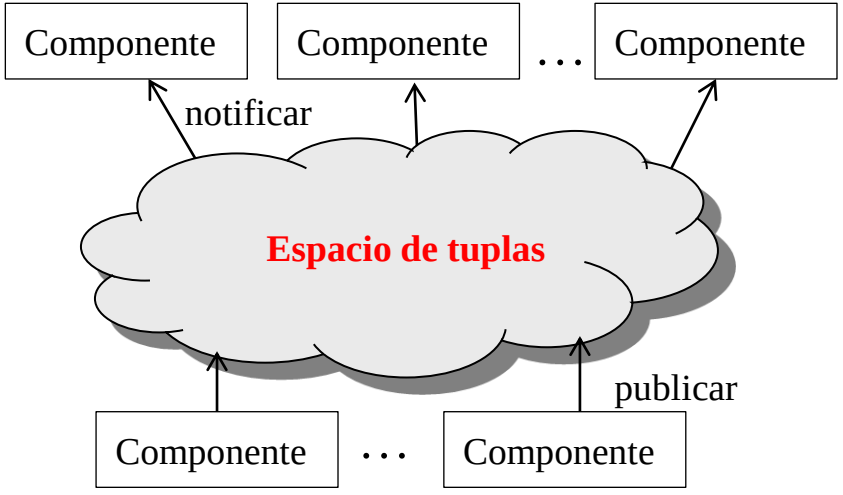
- Responsable de la comunicación entre los procesos
- Desde el punto de vista de la notificación, podemos encontrar dos tipos de buses:
  - Notifican el evento conjuntamente con la información enviada a los suscriptores
  - Notifican el evento y, posteriormente, el suscriptor debe acceder a la información enviada
    - operación de lectura + capacidad de almacenamiento + gestión de datos
- *Granularidad de los eventos*: simples vs. compuestos
  - Eficiencia y escalabilidad del bus

Una primera reflexión

- Con lo visto en clase, ¿cómo podríamos implementar este modelo de sistema?

Una primera reflexión

- Con lo visto en clase, ¿cómo podríamos implementar este modelo de sistema?



- ¿Qué cuestiones deberíamos considerar?

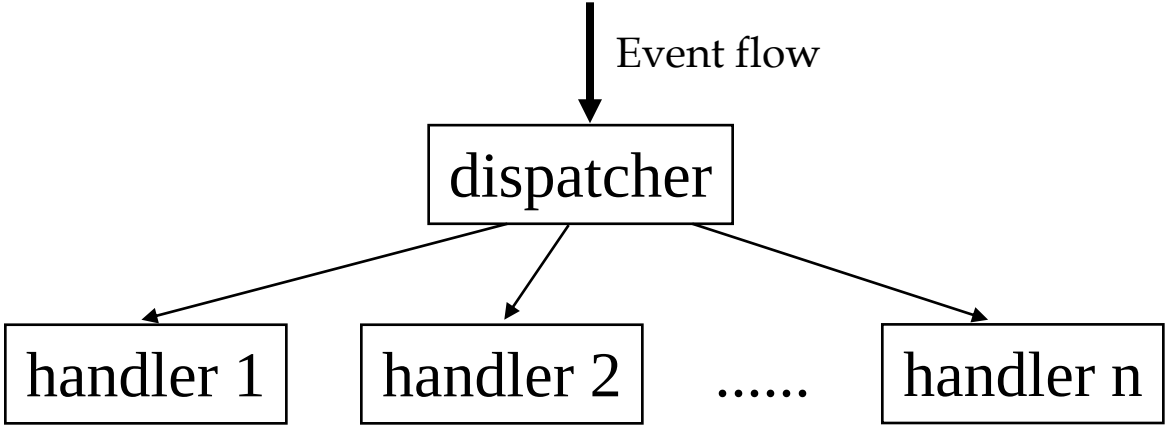
Programación basada en eventos

- Esta idea es aplicable a diferentes niveles de abstracción:
  - Sistemas complejos: controladores de vuelo, sistemas TR, etc.
  - Interfaces de usuario (GUI)
  - Programas dirigidos por eventos

- Surge el paradigma de la "*programación basada en eventos*"
  - el flujo de control de un programa viene determinado por las respuestas del sistema al flujo de "eventos"
- Existen *patrones* que determinan cómo programar basado en eventos
  - Handler, Observer, Chain of responsability, Front controller, etc

Patrón Handler

- Estructura más comúnmente utilizada para implementar una solución dirigida por eventos
- Dos tipos de participantes: "*Dispatcher*" y "*Handlers*"



Patrón Handler

- Bucle del “Dispatcher"

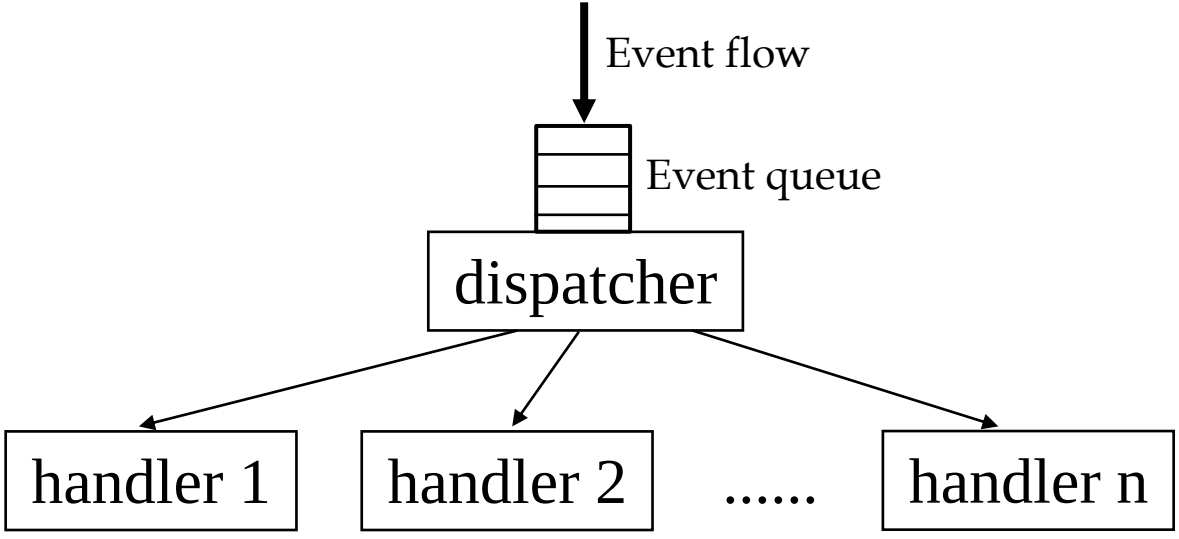
```
loop forever

   e := get next event from the input

   if e = evento_tipo_1
      ejecuta handler_1
   else if e = evento_tipo_2
      ejecuta handler_2
   else if ...
      ...
   else if e es desconocido
      ignorar o enviar excepción
```

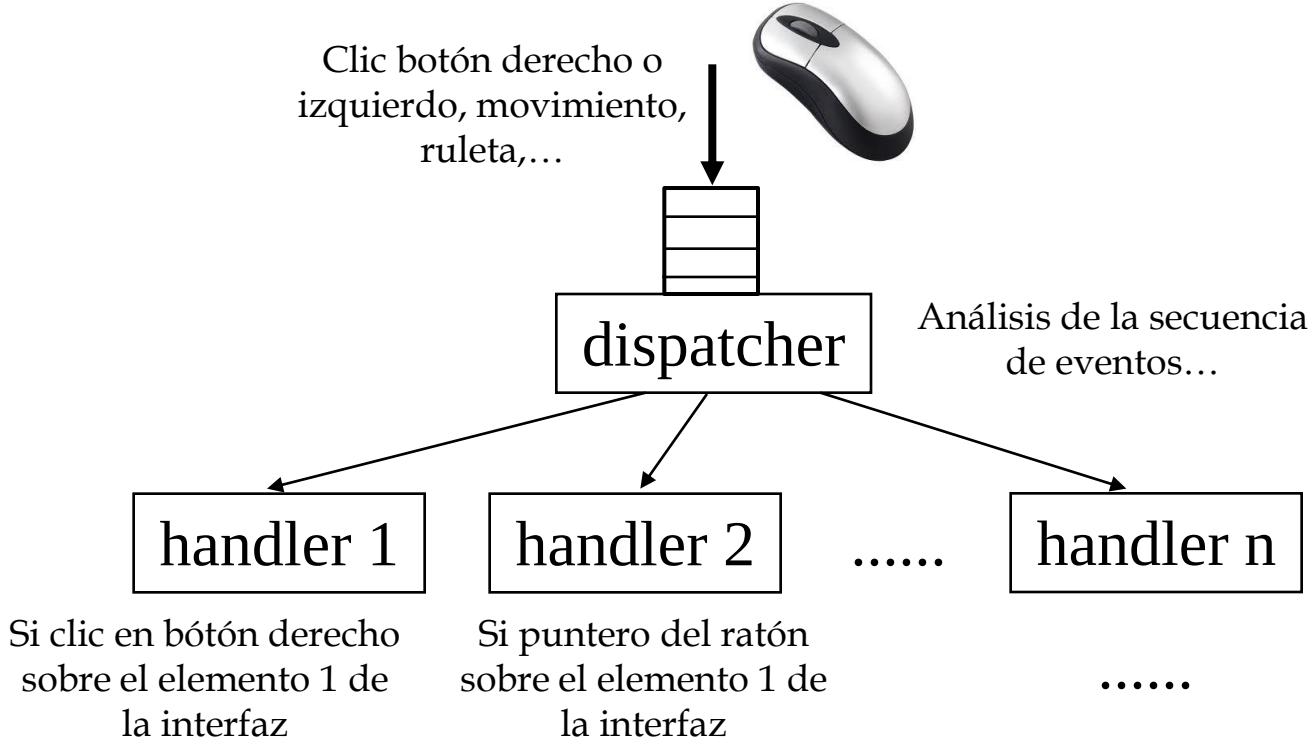
Patrón Handler extendido

- Si el "dispatcher" o los "handlers" no son capaces de gestionar los eventos tan rápido conforme estos están llegado…



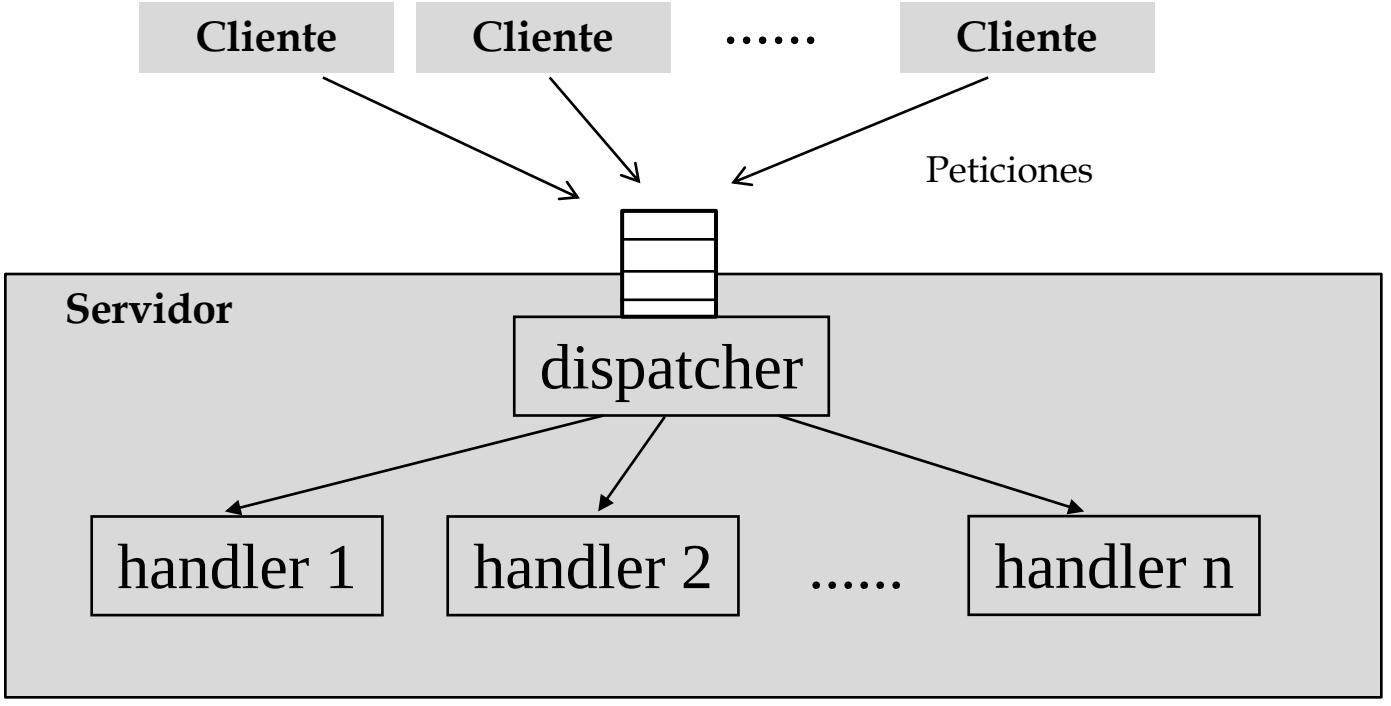
Aplicaciones del patrón Handler

- Implementación de una Interfaz Gráfica de Usuario (GUI)



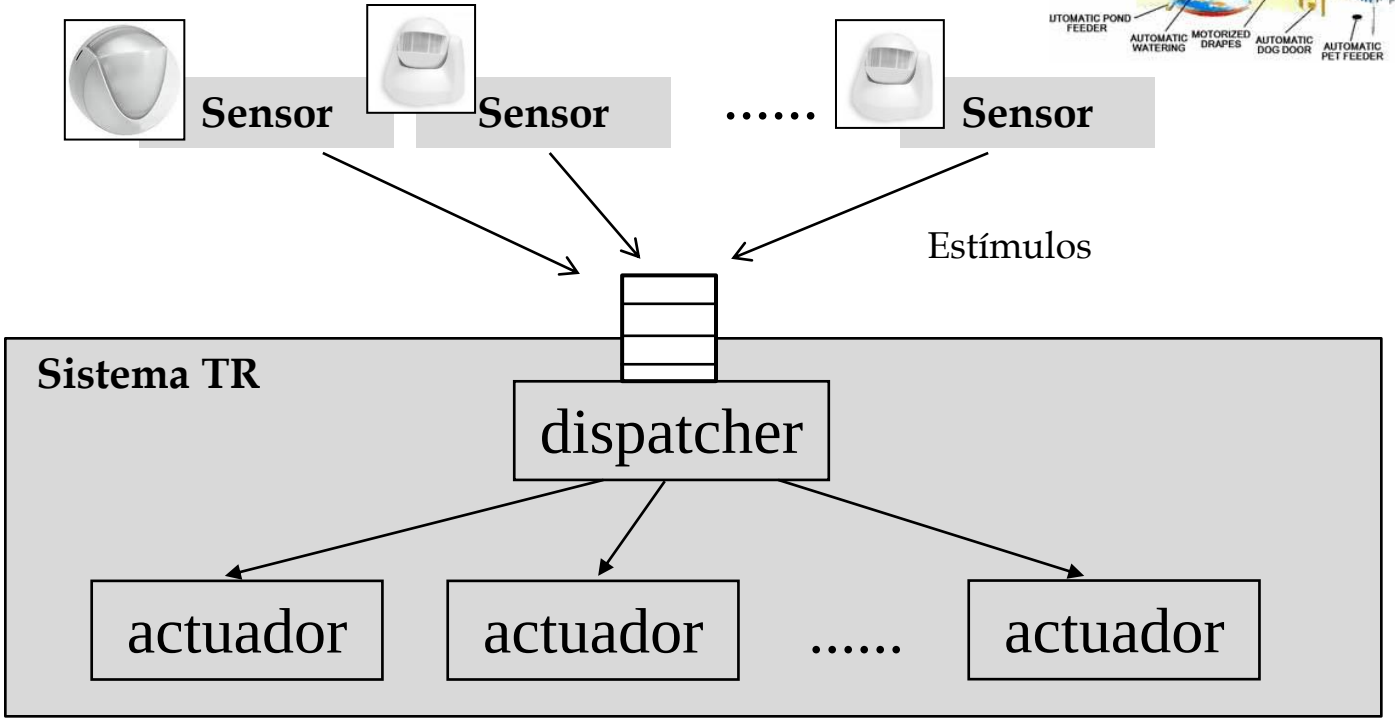
Aplicaciones del patrón Handler

- Implementación de un sistema Cliente/Servidor



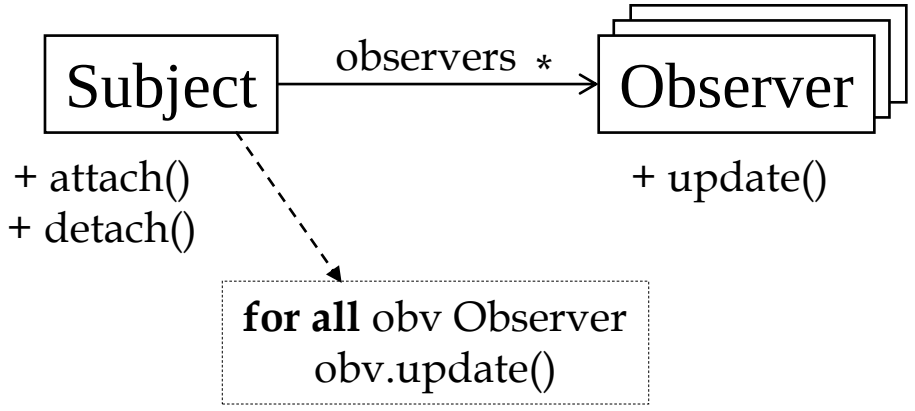
Aplicaciones del patrón Handler

- Implementación de un sistema TR (domótica)



Patrón Observer

- Dos tipos de participantes: “Subject” y “Observer”
- Cambios de estado en el “Subject” provocan notificaciones a los “Observer” previamente registrados

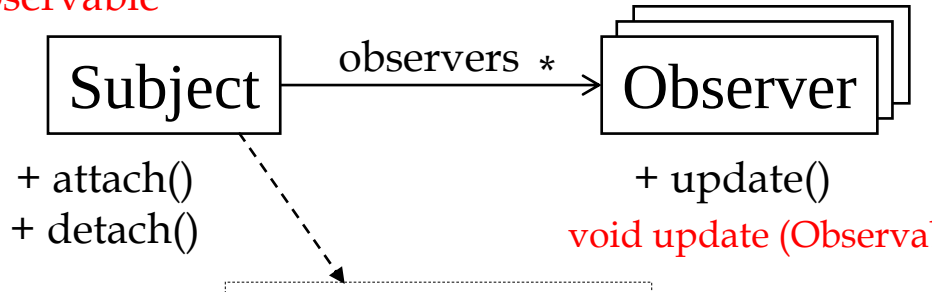


Patrón Observer

- Dos tipos de participantes: "Subject" y "Observer"
- Cambios de estado en el "Subject" provocan notificaciones a los "Observer" previamente registrados

Interface java.util.Observer

java.util.Observable



addObserver(o Observer)
deleteObserver(o Observer)
notifyObserver()
notifyObservers(Object data)

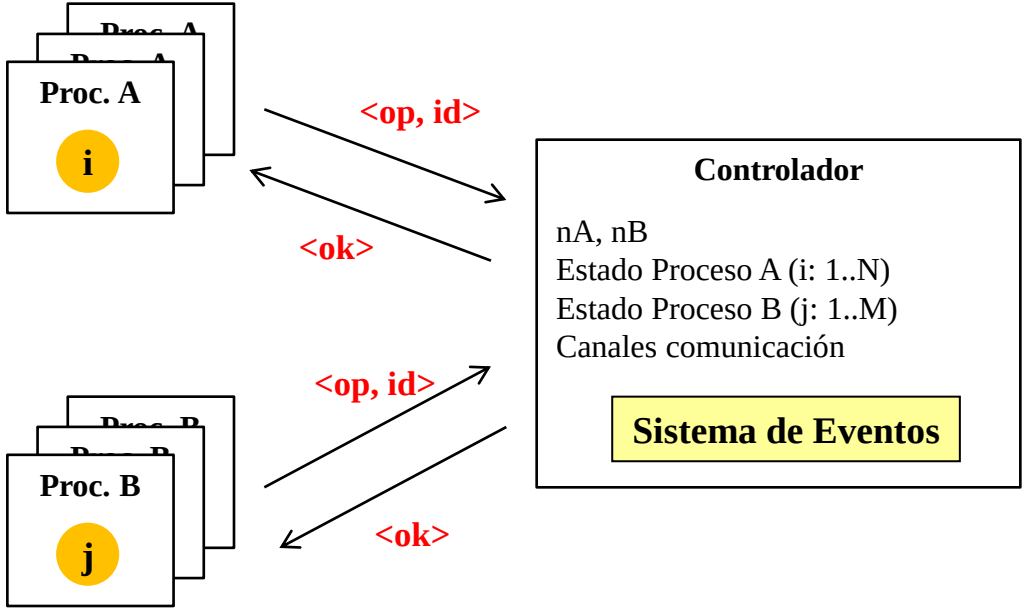
Ejercicio

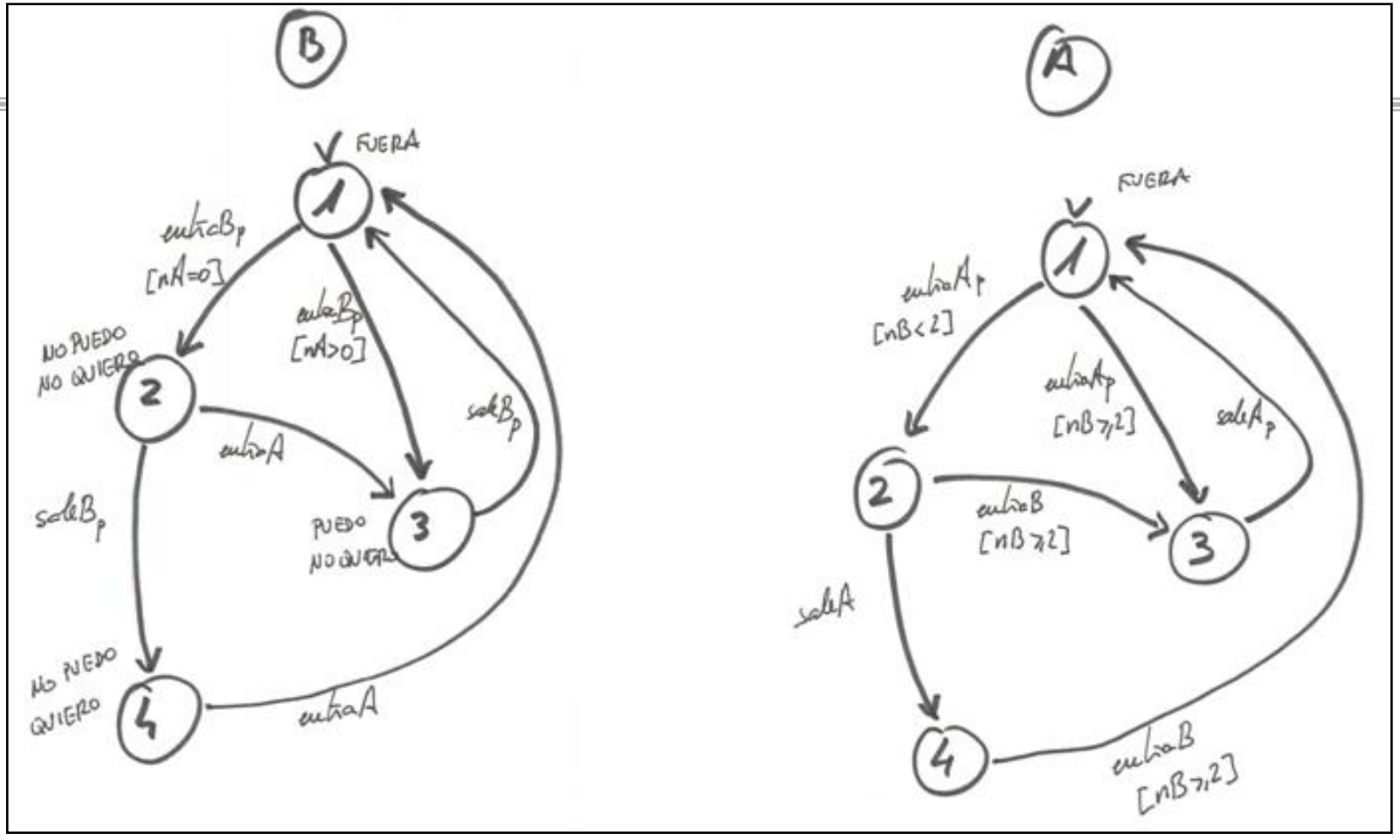
- **Un ejemplo**: arquitectura cliente servidor con estado
  - Dos tipos de procesos, A y B, pueden entrar y salir de una habitación común. Un proceso A no puede salir de la habitación hasta que haya coincidido en ella, simultáneamente, con dos B, mientras que un B no puede salir hasta que haya coincido con al menos un A. Se pide desarrollar el controlador de acceso a la habitación, así como el código de los procesos de tipo A y B. La comunicación de los procesos con el controlador debe hacerse mediante paso asíncrono de mensajes. Asumiendo que siempre hay procesos de ambos tipos queriendo entrar, comentar aspectos como posibilidades de bloqueo, de equidad, etc. en la solución propuesta.

Esquema de solución

Process A (i:1..N)
  loop forever
    send(…, controlador, "entrarA", i)
    …
    send(…, controlador, "salirA", i)
    receive(…, controlador, "ok")

Process B (j:1..M)
  loop forever
    send(…, controlador, "entrarB", j)
    …
    send(…, controlador, "salirB", j)
    receive(…, controlador, "ok")




B
FUERA
1
entraB_p
[nA=0]
entraB_p
[nA>0]
NO PUEDO
NO QUIERO
2
entraA
sale B_p
sale B_p
3
PUEDO
NO QUIERO
NO PUEDO
QUIERO
4
entraA
A
FUERA
1
entraA_p
[nB<2]
entraA_p
[nB≥2]
sale A_p
2
entraB
[nB≥2]
3
saleA
4
entraB
[nB≥2]

Sistema de eventos del controlador

```
process Controlador
   loop forever
      receive(…,eventType,source)

      if (eventType="entrarB")
         nB := nB+1;
         //Actualizo estado del proceso generador del evento
         if (estadoB(source)=1 AND nA>0)
            estadoB(source) := 4
         else if (estadoB(source)=1 AND nA=0)
            estadoB(source):= 2;
           //Actualizo estado de los procesos a los que afecta
         for all N process A
            if (estadoA(j)=2 AND nB>=2)
               estadoA(j):= 4
            else if (estadoA(j)=3 AND nB>=2)
               estadoA(j):= 5;
               send(…,j,"ok")
      else if (eventType="salirB")
         …
```

Handle