Patrones de diseño en Sist Distribuidos (S.D.)

TSR 2018-19 Juansa Sendra

adaptado de '10 common software architecture patterns ...'

Introdución

- Los objetivos al diseñar un S.D. son mejorar **escalabilidad** y **disponibilidad**
  - Para conseguirlos recurrimos a aplicar replicación y técnicas de tolerancia a fallos
- **Patrón de diseño** = Solución sofware reutilizable útil para resolver un problema frecuente
  - Muchas propuestas, nos centramos en las indispensables

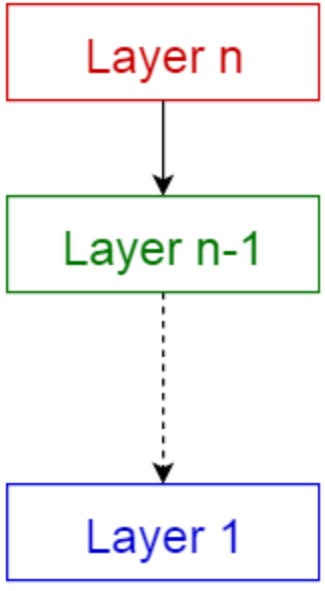
Patrones de diseño para S.D.

- Layered pattern (descomposición por niveles)
- **Client/server pattern**
- **Master/Slave pattern**
- **Pipe/Filter pattern**
- **Broker pattern**
- Peer-to-Peer pattern
- **Event-Bus pattern**
- MVC pattern (modelo/vista/controlador)
- Blackboard pattern
- Interpreter pattern

```
Los patrones en negrita se desarrollan en la asignatura
```

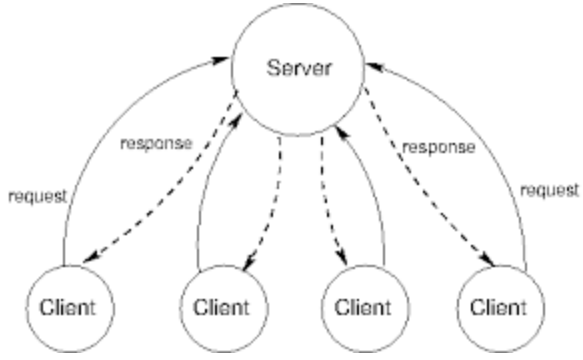
Layered pattern

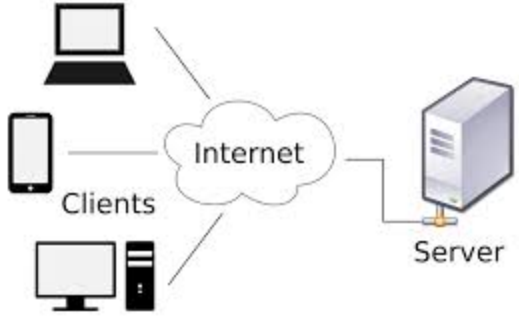
- Descomposición en subtareas, de forma que:
  - Cada subtarea (nivel) a $\neq$ nivel de abstracción
  - El nivel n proporciona servicios al n+1
- Son frecuentes 4 niveles:
  - Presentación (interfaz usuario)
  - Aplicación (servicio)
  - Lógica del negocio (dominio)
  - Acceso a datos (persistencia)
- Útil para aplicaciones de escritorio, e-commerce, ..


Layer n
Layer n-1
Layer 1

Cliente/Servidor

- Un Servidor, que proporciona servicios bajo demanda a múltiples clientes
- N Clientes, que solicitan servicios al servidor
- Sigue un modelo de petición/respuesta sincrónico
- Tras responder a un cliente, un servidor atiende (escucha) nuevas peticiones de otros clientes
    - El servidor puede utilizar concurrencia interna para atender de forma simultánea a N clientes<p
- Útil para aplicaciones online (e-mail, compartición de documentos, aplicaciones bancarias, ...)


Server
response
response
request
request
Client
Client
Client
Client

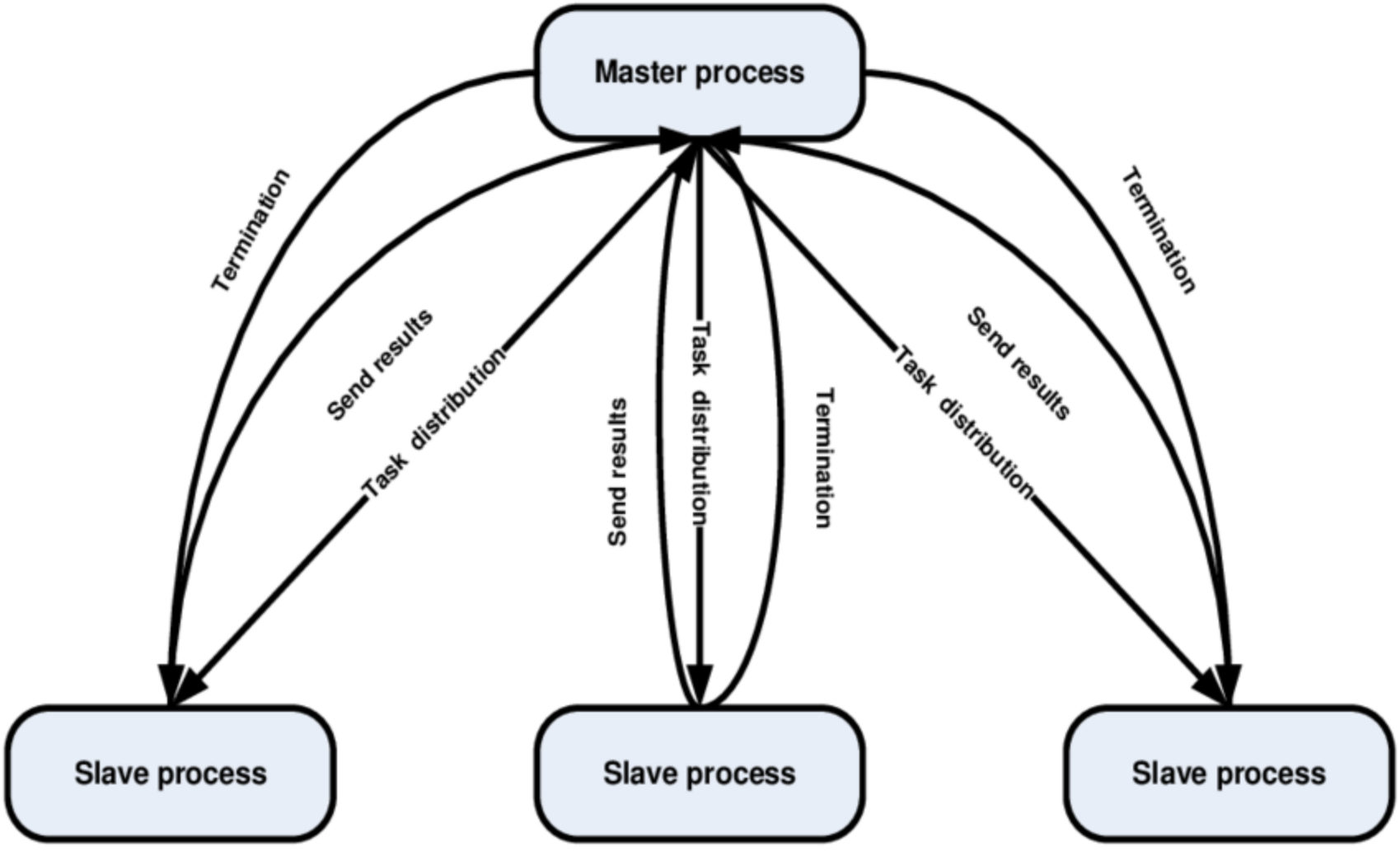

Internet
Clients
Server

Cliente/Servidor.- ventajas/inconvenientes

- ventajas
  - Bueno para modelar un conjunto de servicios accesibles a los clientes
  - Simple si operaciones idempotentes (servidores stateless)
  - Amplia difusión
- inconvenientes
  - Complejo si se necesitan servicios con estado
  - Los servidores pueden convertirse en puntos de fallo único

Maestro/Esclavo

- El maestro distribuye trabajo entre distintos componentes esclavo equivalentes entre sí
- Cada esclavo completa su parte y devuelve el resultado al maestro
- El maestro computa la respuesta final a partir de los resultados parciales devueltos por los esclavos
- Útil para:
    - escalar tareas que pueden dividirse en fragmentos equivalentes (ej. aplicar una misma tarea sobre fragmentos distintos de los datos)
    - replicación pasiva

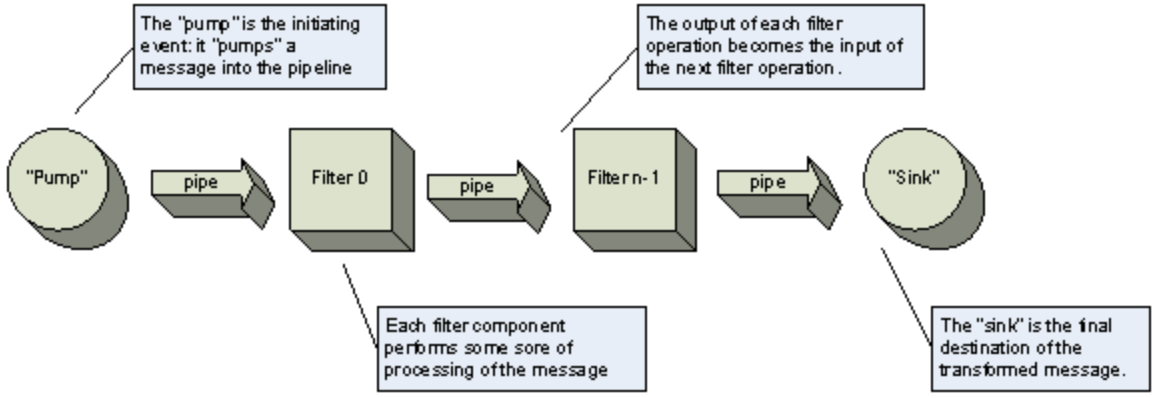

Master process
Termination
Send results
Task distribution
Send results
Task distribution
Termination
Task distribution
Send results
Termination
Slave process
Slave process
Slave process

Maestro/esclavo.- ventajas/inconvenientes

- ventajas
  - simple
  - escalable
- inconvenientes
  - Sólo aplicable a problemas con posibilidad de descomposición en fragmentos equivalentes e independientes entre sí
  - Los esclavos están aislados (no comparten estado)
  - La comunicación maestro/esclavos/maestro puede introducir lantencias

Pipe/Filter

- Sistemas que producen/procesan un flujo de datos
- Cada etapa de procesamiento corresponde a un componente Filter
- Filter conectados entre sí mediante tuberías (Pipe)
    - los datos a procesar atraviesan las tuberías
    - las tuberías pueden usarse para buffering y/o sincronización
- Útil para compilación distribuida (análisis léxico, parsing, análisis semántico, generación código), workflow en biotecnología, ...

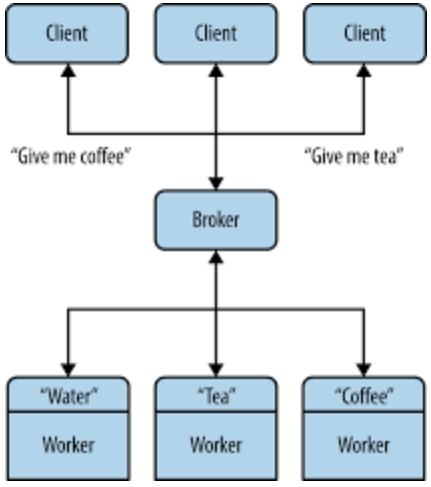

The "pump" is the initiating event: it "pumps" a message into the pipeline
The output of each filter operation becomes the input of the next filter operation.
"Pump"
pipe
Filter 0
pipe
Filter n-1
pipe
"Sink"
Each filter component performs some sore of processing of the message
The "sink" is the final destination of the transformed message.

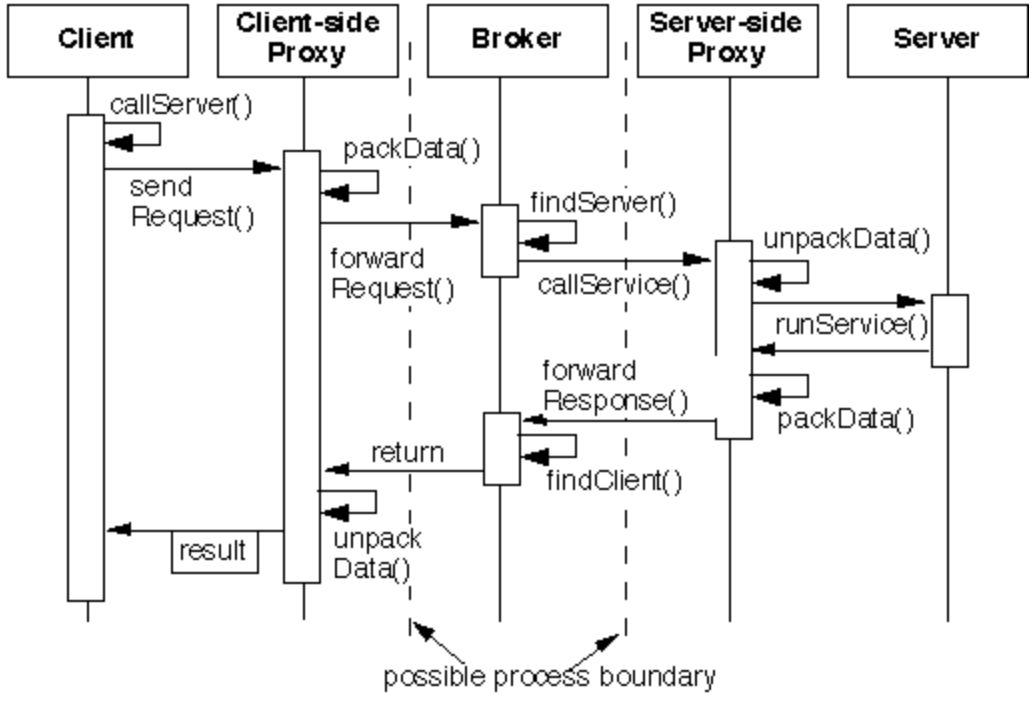
Pipe/Filter.- ventajas/inconvenientes

- ventajas
  - muy flexible. Es fácil añadir/modificar filtros
  - Filtros reutilizables, y combinables en distintas configuraciones
  - refleja la concurrencia implícita en los problemas basados en flujos
- inconvenientes
  - El filtro más lento limita el rendimiento global
  - sobrecarga por el desplazmiento de los datos entre filtros

Broker

- Patrón usado para estructurar SD con componentes desacoplados
    - Los componentes interactúan entre sí mediante invocación de servicios remotos
- El componente broker (intermediario) se ocupa de la coordinación/comunicación entre componentes
    - Los servidores comunican al broker sus características
    - El cliente solicita los servicios al broker
    - El broker redirecciona cada solicitud al servidor adecuado (ej. equilibrando la carga)


Client
Client
Client
"Give me coffee"
"Give me tea"
Broker
"Water"
Worker
"Tea"
Worker
"Coffee"
Worker

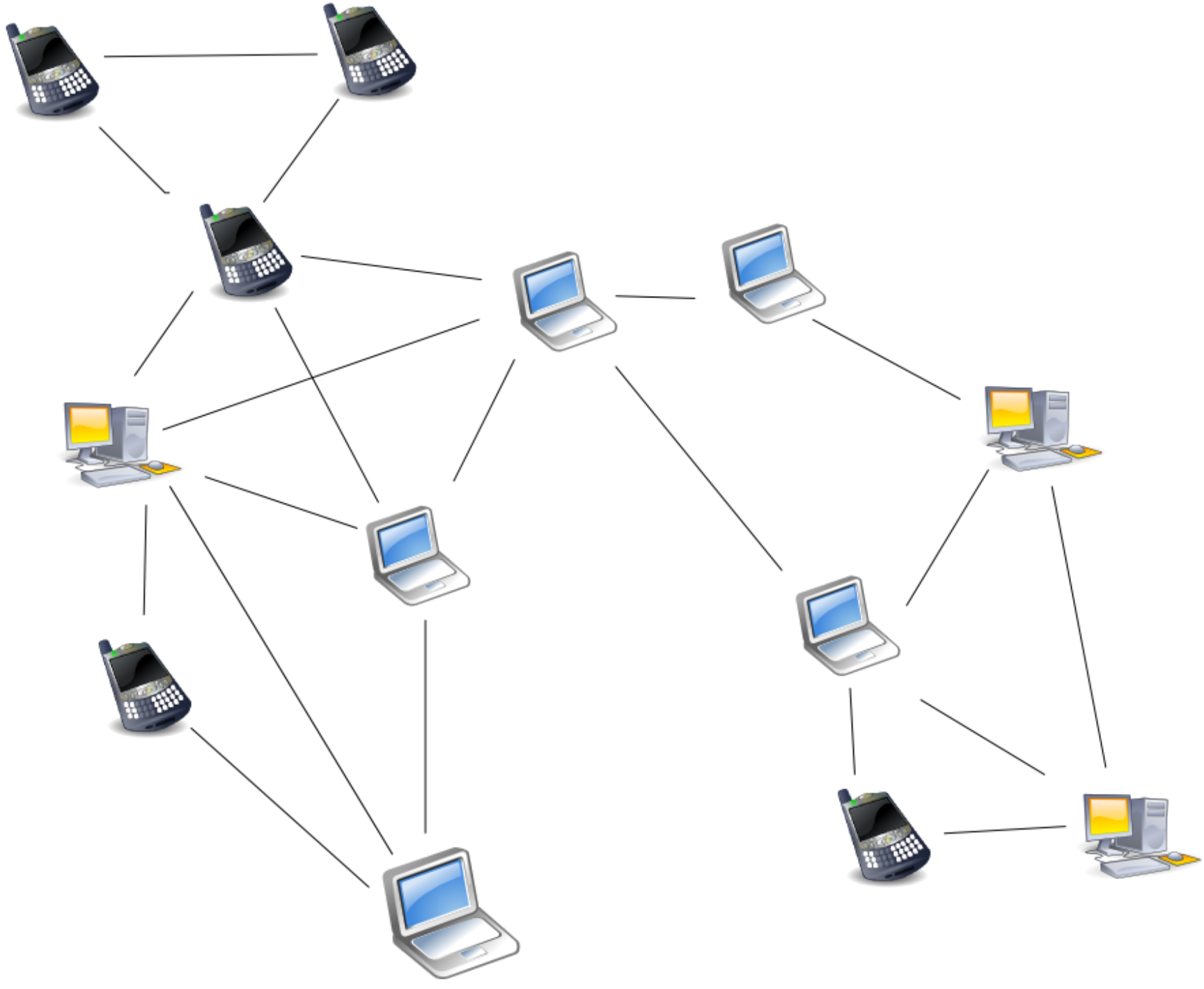

Client
Client-side Proxy
Broker
Server-side Proxy
Server
callServer()
packData()
send Request()
findServer()
unpackData()
forward Request()
callService()
runService()
forward Response()
packData()
return
findClient()
result
unpack Data()
possible process boundary

Broker.- ventajas/inconvenientes

- ventajas
  - distribución transparente al cliente y al desarrollador
  - permite transparencia de fallos
  - facilita la contabilidad asociada al uso que hacen los clientes del sistema, proporciona un nivel de aislamiento (seguridad), permite transformar las peticiones según las necesidades del sistema, etc.
- inconvenientes
  - requiere estandarización de las descripciones de servicios
  - si se añaden políticas de equilibrado de carga y de reconfiguración ante fallos, el broker se complica bastante
  - requiere broker de respaldo para evitar punto de fallo único

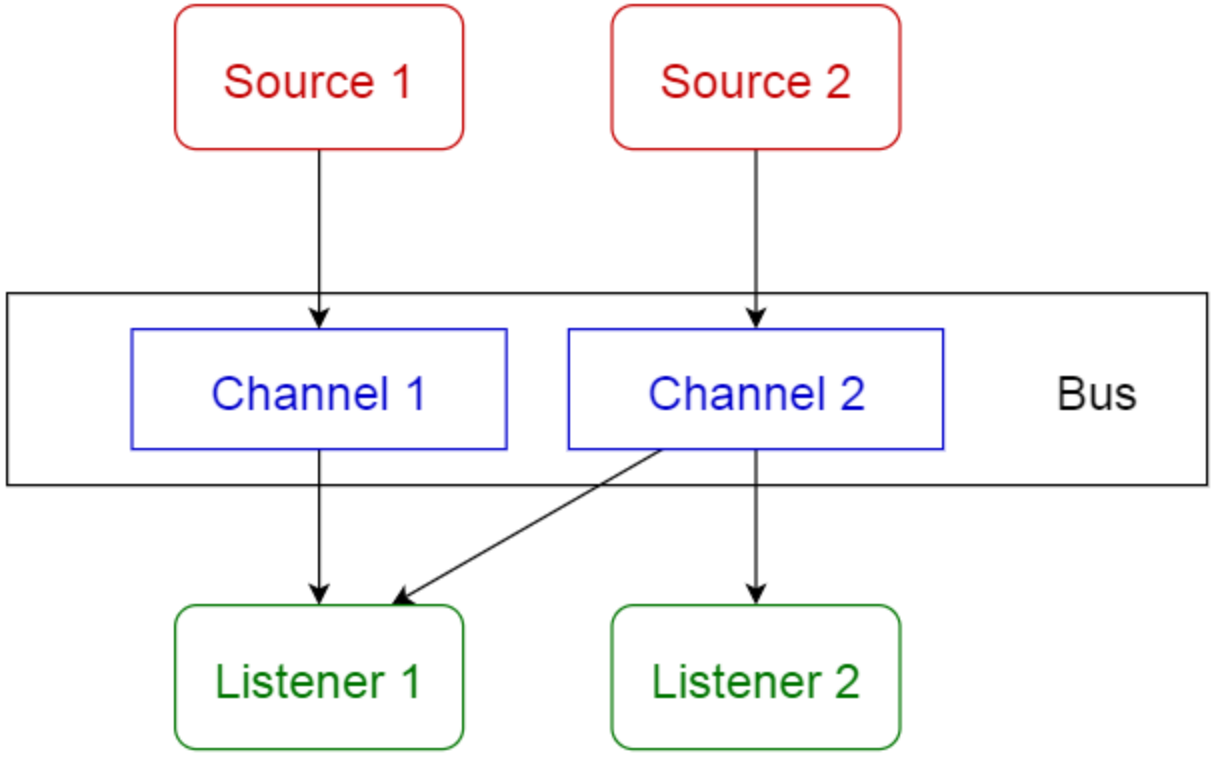
Peer-to-Peer

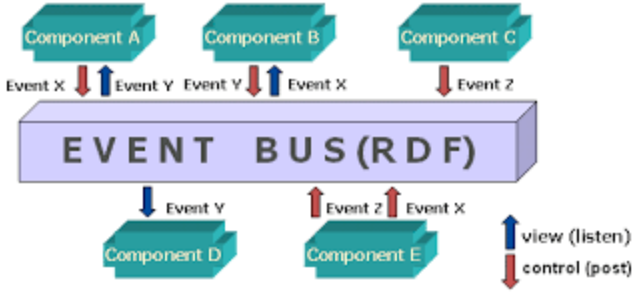
- Los componentes (Peers) pueden actuar como:
  - clientes (solicitan servicios a otros peers)
  - servidores (proporcionan servicios a otros peers)
- Un Peer puede actuar como cliente, como servidor, o como ambos
  - y puede cambiar su rol de forma dinámica
- Suele ser una red no estructurada, con nodos heterogéneos y volátiles (se conectan/desconectan en momentos impredecibles)
- Útil para aplicaciones no críticas que requieren replicación masiva (redes de compartición de ficheros, protocolos multimedia como P2PTV, ..)



Event-Bus

- Trata con eventos
- Componentes principales
  - fuente (emisor) de eventos.- publica mensajes a canales concretos o a un bus de eventos
  - oyente (listener) de eventos.- se suscribe a canales concretos (reciben notificaciones ante la llegada de mensajes a esos canales)
  - canales y/o bus de eventos
- Útil para servicios de notificación, desarrollo Android, ...


Source 1
Source 2
Channel 1
Channel 2
Bus
Listener 1
Listener 2

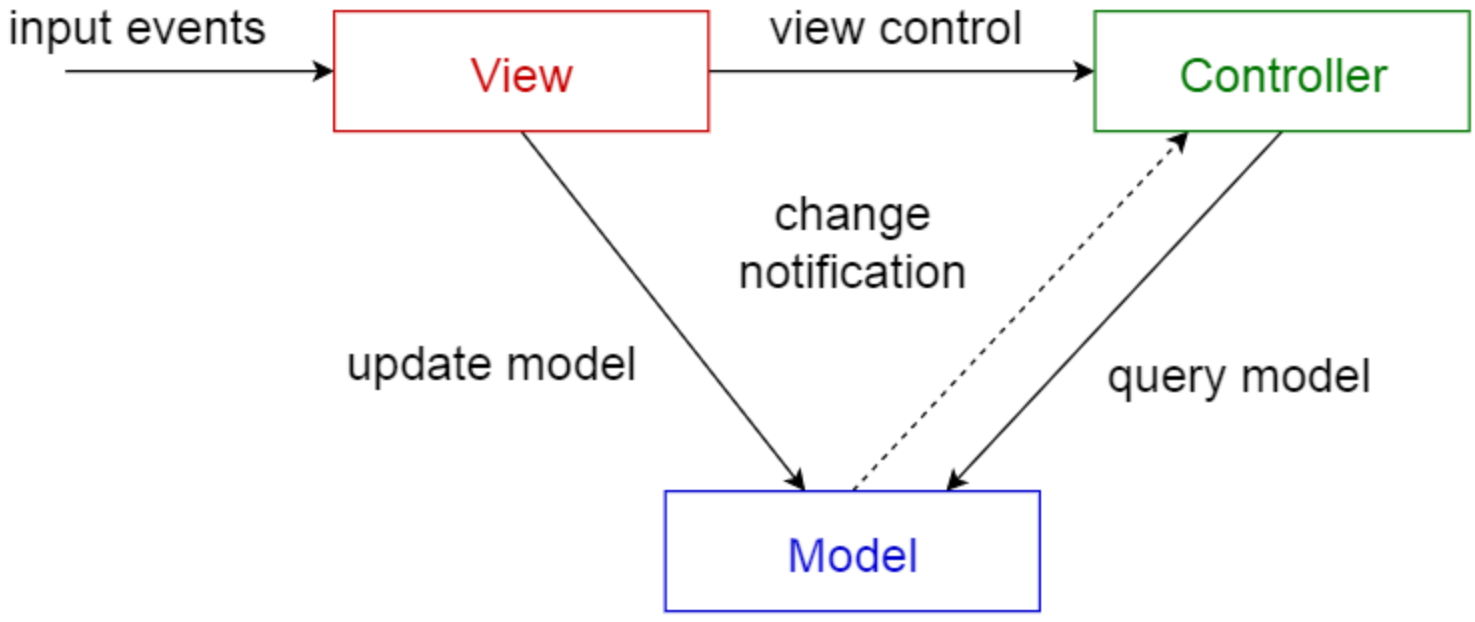

Component A
Component B
Component C
Event X
Event Y
Event Y
Event X
Event Z
EVENT BUS(RDF)
Event Y
Event Z
Event X
Component D
Component E
view (listen)
control (post)

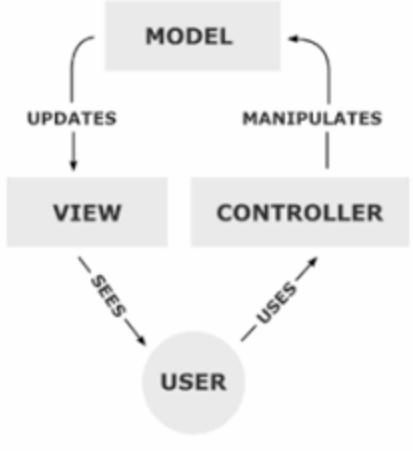
Event-bus.- ventajas/inconvenientes

- ventajas
  - Flexible.- es fácil añadir publicadores, suscriptores y conexiones
  - Adecuado para aplicaciones masivamente distribuidas
- inconvenientes
  - El bus puede convertirse en un cuello de botella

MVC (modelo/vista/controlador)

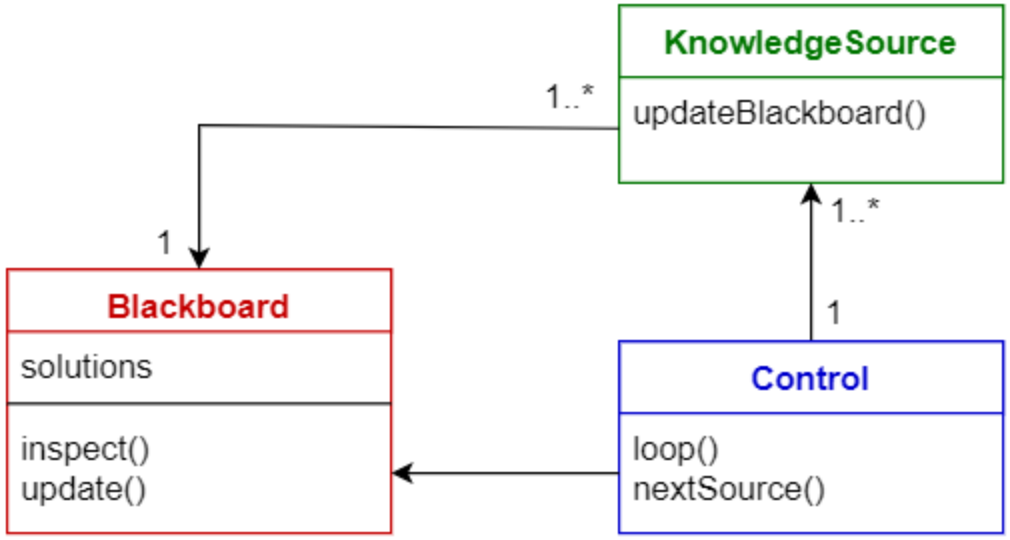
- Útil para aplicaciones interactivas. Las divide en:
  - modelo.- contiene la funcionalidad y datos básicos
  - vista.- muestra la información al usuario (puede definir N vistas sobre un mismo modelo, posiblemente presentando los datos de forma distinta)
  - controlador.- Gestiona los enventos de entrada del usuario
- Esta división separa la representación de la información de la forma en que se presenta al usuario
- Mayor desacoplamiento $\rightarrow$ mejor reutilización

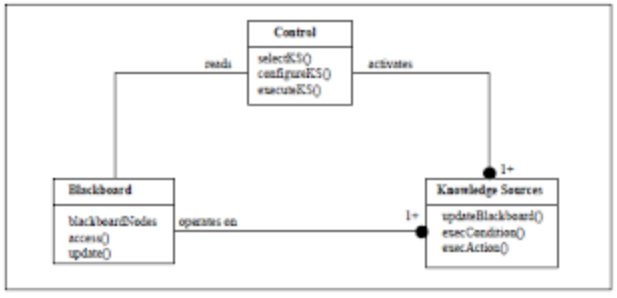

input events
View
view control
Controller
change
notification
update model
query model
Model


MODEL
UPDATES
MANIPULATES
VIEW
CONTROLLER
SEES
USES
USER

Blackboard (pizarra)

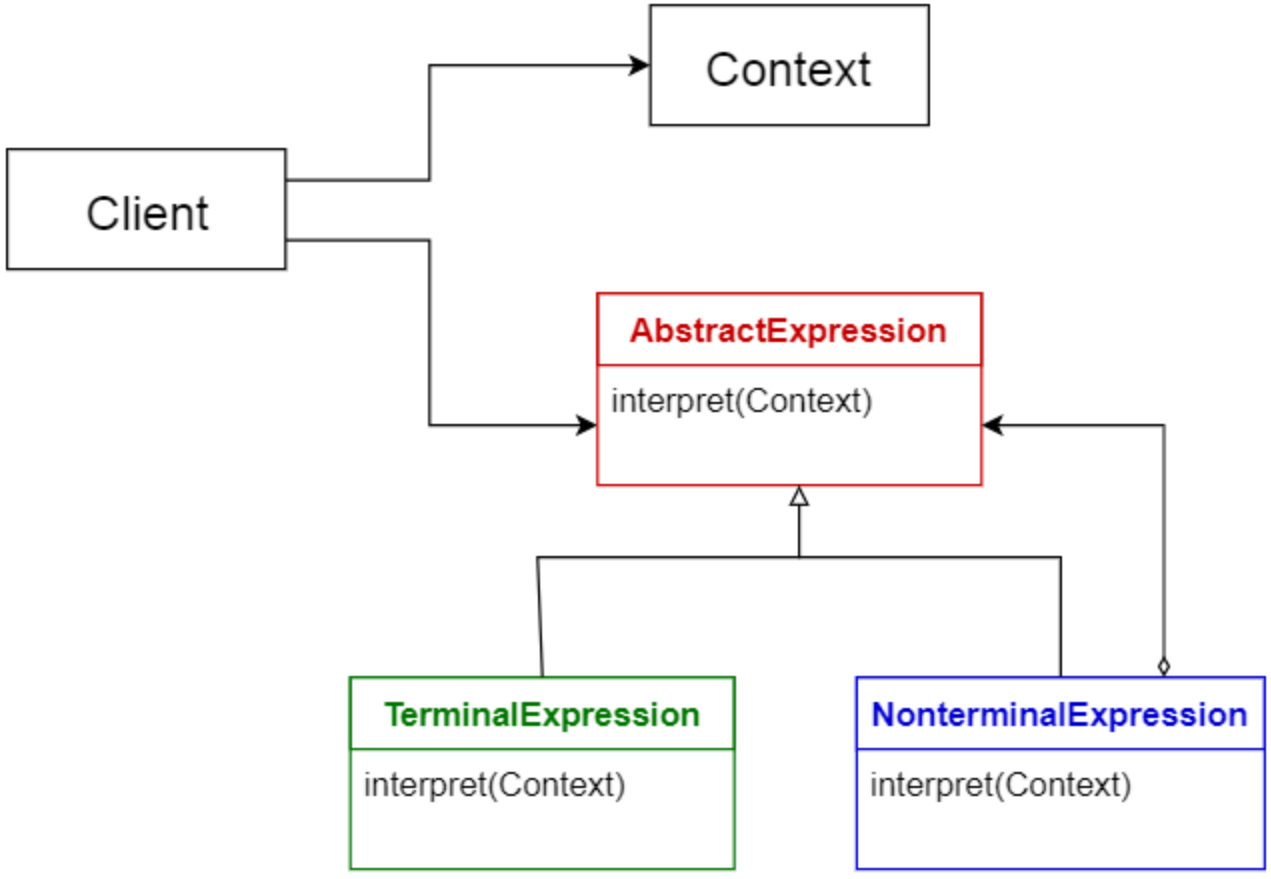
- Resulta útil para problemas en los que no se conoce una estrategia de solución determinista
    - reconocimiento de formas, identificación y seguimiento de vehículos, identificación de la estructura de proteínas, ...
- Componentes:
    - pizarra.- Representa una memoria global estructurada que contiene objetos del espacio de solución
    - fuente de conocimiento.- módulos especializados, con su propia representación
    - Control.- selecciona, configura y ejecuta módulos
- Todos los componentes tienen acceso al blackboard
    - Pueden producir nuevos objetos y añadirlos a la pizarra
    - O buscar datos en la pizarra utilizando pattern-matching con la fuente de conocimiento existente


KnowledgeSource
updateBlackboard()
1..*
1
1..*
1
Blackboard
solutions
inspect()
update()
Control
loop()
nextSource()


Control
selectKS()
configureKS()
executeKS()
reads
activates
1+
Blackboard
blackboardNodes
access()
update()
operates on
1+
Knowledge Sources
updateBlackboard()
execCondition()
execAction()

Interpreter pattern

- Se utiliza para diseñar un componente qeu interpreta programas escritos en un lenguaje dedicado
    - especifica cómo evaluar las expresiones y sentencias de dicho programa
    - la idea básica es disponer de una clase por cada símbolo del lenguaje
- Se utiliza en lenguajes de interrogación (ej SQL) y lenguajes utilizados para describir protocolos de comunicación


Context
Client
AbstractExpression
interpret(Context)
TerminalExpression
interpret(Context)
NonterminalExpression
interpret(Context)