ARQUITECTURA Y DISEÑO DE SISTEMAS

ESTILOS Y PATRONES DE ARQUITECTURA – PARTE 3

ELSA ESTEVEZ
UNIVERSIDAD NACIONAL DEL SUR
DEPARTAMENTO DE CIENCIAS E INGENIERIA DE LA COMPUTACION

REPASO - PATRONES ARQUITECTÓNICOS

DEFINICIÓN

Un patrón arquitectónico es una colección de decisiones de diseño arquitectónicas que tiene un nombre específico y que son aplicables a problemas de diseño recurrentes, y son parametrizadas para tener en cuenta diferentes contextos de desarrollo de software en los cuales el problema aparece.

Taylor, Medvidovic and Dashofy

- provee un conjunto de decisiones específicas de diseño que han sido identificadas como efectivas para organizar ciertas clases de sistemas de software o, más típicamente, subsistemas específicos.

- estas decisiones de diseño pueden pensarse como “configurables”, ya que necesitan ser instanciadas con los componentes y conectores particulares a una aplicación.

REPASO - ESTILOS Y PATRONES - DIFERENCIAS

	ESTILO	PATRÓN
Alcance	Aplican a un contexto de desarrollo: ○ “sistemas altamente distribuidos”, ○ “sistemas intensivos en GUI”	Aplican a problemas de diseño específicos: ○ El estado del sistema debe presentarse de múltiples formas ○ La lógica de negocio debe estar separada del acceso a datos
Abstracción	Son muy abstractos para producir un diseño concreto del sistema.	Son fragmentos arquitectónicos parametrizados que pueden ser pensados como una pieza concreta de diseño.
Relación	Un sistema diseñado de acuerdo a las reglas de un único estilo puede involucrar el uso de múltiples patrones	Un único patrón puede ser aplicado a sistemas diseñados de acuerdo a los lineamientos de múltiples estilos

CONTENIDO

CAPAS (LAYERS) Y TIERS – CAPAS

- Una aplicación de N-capas eventualmente puede estar físicamente dispuesta en una misma computadora física (un único tier)
- Los componentes de distintos niveles (capas) se comunican a través de interfaces bien definidas, y respetando las restricciones del estilo.
- La comunicación entre niveles es explícita y levemente acoplada.
- Una arquitectura por niveles agrupa funcionalidad en un mismo nivel siguiendo algún criterio.
- Principales ventajas de un estilo arquitectónico por niveles son:
  - ✓ abstracción
  - ✓ separación de intereses
  - ✓ reusabilidad
  - ✓ testeabilidad

CAPAS (LAYERS) Y TIERS – TIERS

- Una arquitectura N-tier generalmente tiene al menos N= tres partes lógicas separadas, cada una ubicada sobre servidores físicamente distintos.

- Cada tier es responsable de funcionalidad especifica.

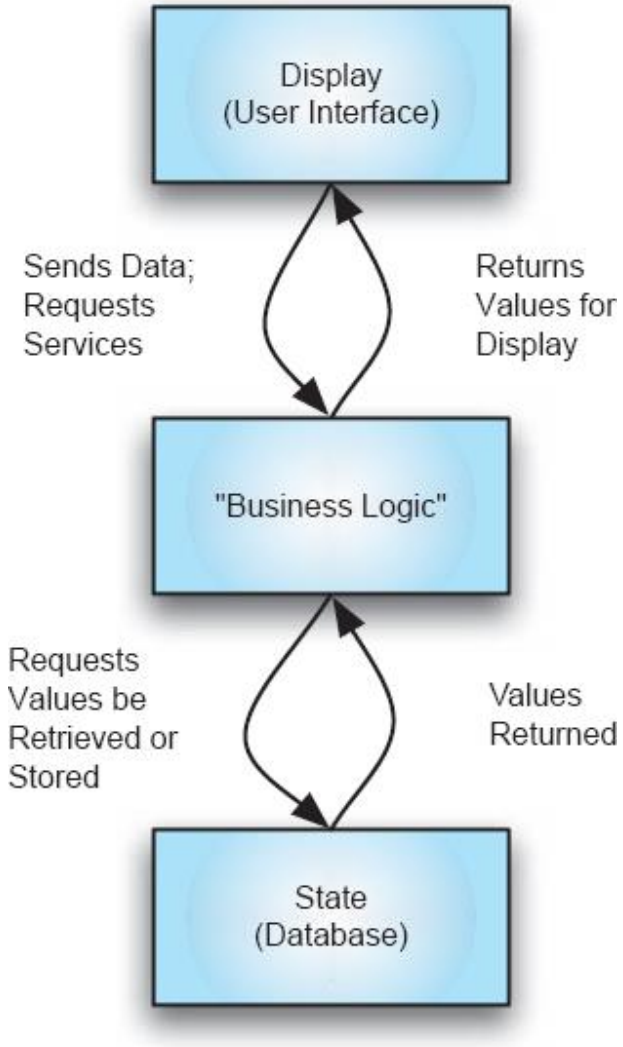
- Cada tier es “independiente”. La comunicación entre tiers típicamente es asincrónica.

- Principales beneficios de la distribución en tiers:
  - ✓ mantenibilidad
  - ✓ escalabilidad
  - ✓ flexibilidad
  - ✓ disponibilidad

1) PATRÓN: ESTADO-LÓGICA-PRESENTACIÓN (3 TIERS)

- Comúnmente utilizado en aplicaciones empresariales donde existe
  - ✓ Un almacenamiento de datos detrás de las reglas de la lógica de negocio
  - ✓ La lógica de negocio es accedida por los componentes de la interfaz de usuario

- Ejemplos:
  - ✓ Aplicaciones empresariales
  - ✓ Juegos multi-player
  - ✓ Aplicaciones web



2) PATRÓN: MODEL-VIEW-CONTROLLER (MVC)

Contexto: En las aplicaciones “on-line” el software que maneja la interface requiere modificaciones con mayor frecuencia.

Problema: Cómo mantener separadas la funcionalidad que corresponde a la interfaz, de la funcionalidad de la aplicación; y sin embargo, no dejar de responder a la interacción del usuario o a los cambios en los datos de la aplicación.

Solución: el patrón Model-View-Controler, que separa la funcionalidad de la aplicación en tres clases de componentes:

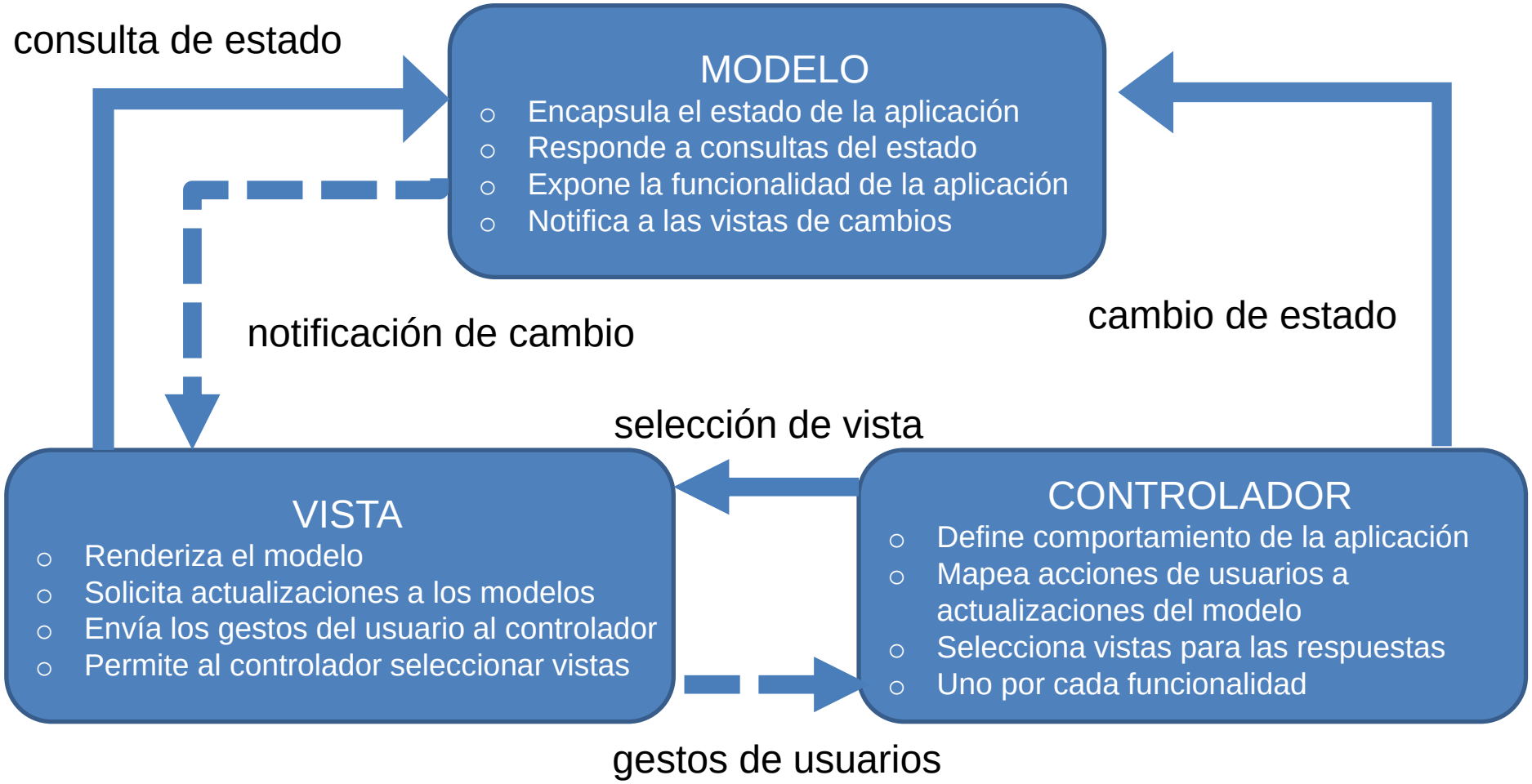
- ✓ Un modelo, que contiene los datos.
- ✓ Una vista, que muestra una “porción” de los datos subyacentes e interactúa con el controlador
- ✓ Un controlador, que medie entre ambos y atienda los eventos.

PATRÓN: MODEL-VIEW-CONTROLLER - COMPONENTES

Separa la funcionalidad de la aplicación en 3 tipos de componentes:

- Modelo: es la representación del estado (datos) de la aplicación y contiene (provee una interfaz) la lógica de la aplicación.
- Vista: es un componente UI que o bien muestra una representación del modelo al usuario y/o permite algún tipo de entrada de usuario
- Controller: maneja la interacción entre el modelo y las vistas, traduciendo las acciones de usuario en cambios al modelo o cambios a la vista.

PATRÓN: MODEL-VIEW-CONTROLLER - COMPONENTES



PATRÓN: MODEL-VIEW-CONTROLLER - RESPONSABILIDADES

- Modelo
  - ✓ Acceder a la capa de almacenamiento de datos
  - ✓ Definir las reglas de negocio
  - ✓ Notificar a las vistas los cambios en los datos (modelo activo)

- Vista
  - ✓ Recibir los eventos de entrada
  - ✓ Contener las reglas de gestión de eventos (si Evento E entonces Acción A, estas acciones pueden suponer peticiones al modelo o la vista)

- Controlador
  - ✓ Recibir los eventos de entrada
  - ✓ Contener las reglas de gestión de eventos (si Evento E entonces Acción A, estas acciones pueden suponer peticiones al modelo o la vista)

PATRÓN: MODEL-VIEW-CONTROLLER - CARACTERISTICAS

Objetivo: Promover la separación de intereses

- ✓ Ayuda a la Testeabilidad
- ✓ Permite caminos de desarrollo independientes.

Historia

- ✓ Es uno de los patrones de diseño más conocidos , teniendo influencia desde su creación (1970) en muchos UI frameworks y en la forma de pensar sobre el diseño de interfaces de usuario.

Ejemplos

- Java Swing, Microsoft ASP.NET MVC, Adobe's Flex

PATRÓN: MODEL-VIEW-CONTROLLER – CUANDO ?

Usarlo si….

- ✓ las separaciones indicadas anteriormente (especialmente la separación de la presentación y el modelo) son útiles.

Evitarlo si…

- ✓ se tiene una aplicación muy simple donde el modelo no tiene comportamiento.
- ✓ las tecnologías a utilizar no brindan la infraestructura necesaria.

SEPARACIÓN 1 – VISTA-MODELO

- es uno de los más importantes principios de diseño
- generalmente tienen intereses distintos
  - Vista – se ocupa por mecanismos de UI y por cómo diseñar una buena UI
  - Modelo – se piensa en términos de reglas de negocio y, quizás, de interacciones con la base de datos.
- se podría querer ver la misma información del modelo de distintas formas
- objetos no visuales son más fáciles de testear que objetos visuales
- la presentación depende del modelo pero el modelo no depende la presentación

SEPARACIÓN 2 – CONTROLADOR – VISTA

- No es tan importante, pero igualmente brinda beneficios. Permitiría tener más de un controlador por vista, o distintas vistas usar el mismo controlador

- Ejemplo: Soportar comportamiento de edición y visualización con una vista

  ✓ Podríamos tener 2 controladores, uno para cada caso, donde los controladores son Strategies (GoF) de la vista.

PATRÓN MVC – ANÁLISIS 1

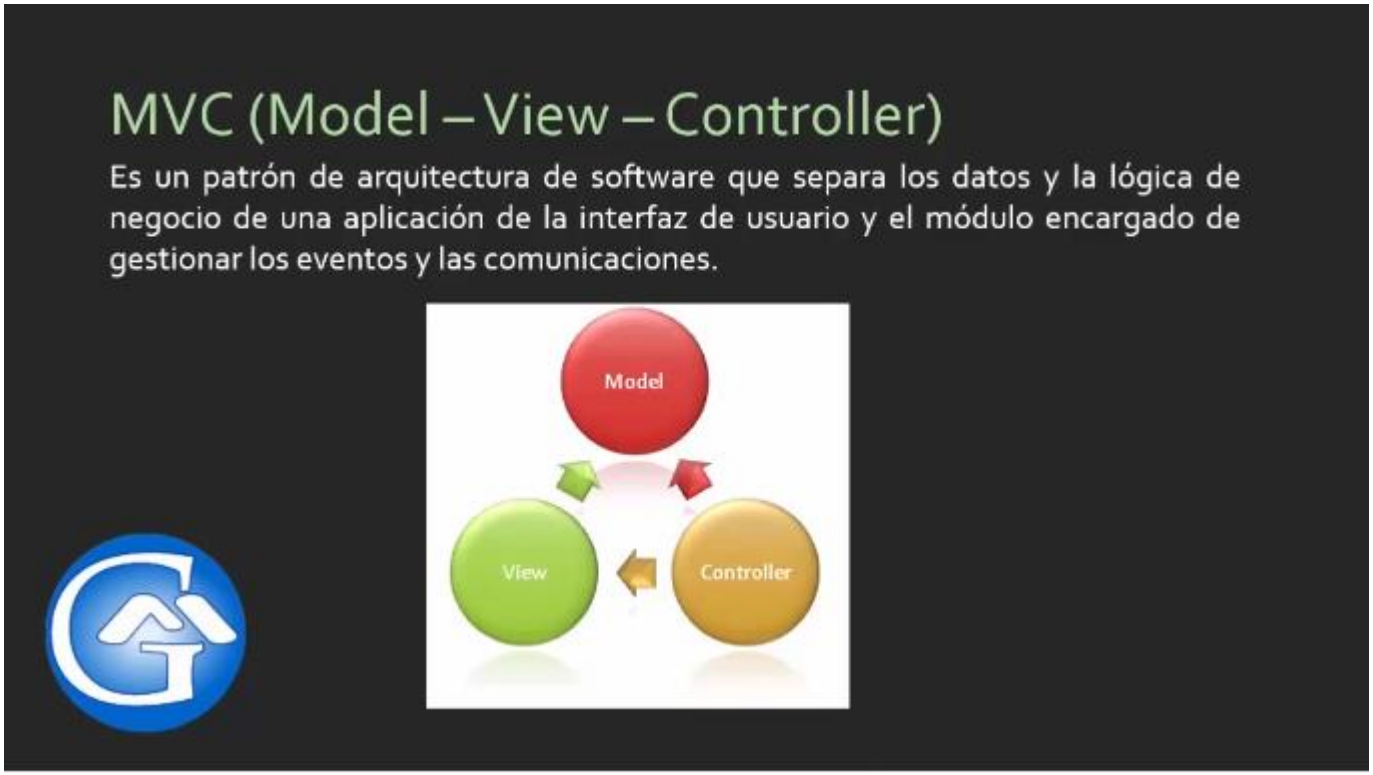
DESCRIPCIÓN	Separa la funcionalidad del sistema entre los componentes modelo, vista y controlador
COMPONENTES	○ Modelo - la representación de los datos o estado de la aplicación. Contiene o (o cuenta con la interfaz) la lógica de la aplicación ○ Vista - es un componente interfaz de usuario. Produce representaciones del modelo para el usuario y/o permite alguna manera de ingresar datos. ○ Controlador - maneja la interacción entre el modelo y la vista, trasladando las acciones de usuario en cambios al modelo o la vista

PATRÓN MVC – ANÁLISIS 2

RELACIONES	La relación notifica conecta instancias del modelo, vista y controlador
RESTRICCIONES	○ debe existir al menos una instancia de vista, modelo y controlador ○ el componente modelo no debe interactuar directamente con el controlador
DEBILIDADES	○ Puede ser demasiado complejo en aplicaciones con interfaces de usuario simples ○ Las abstracciones modelo-vista-controlador pueden no ser adecuadas para algunas herramientas de interface de usuario

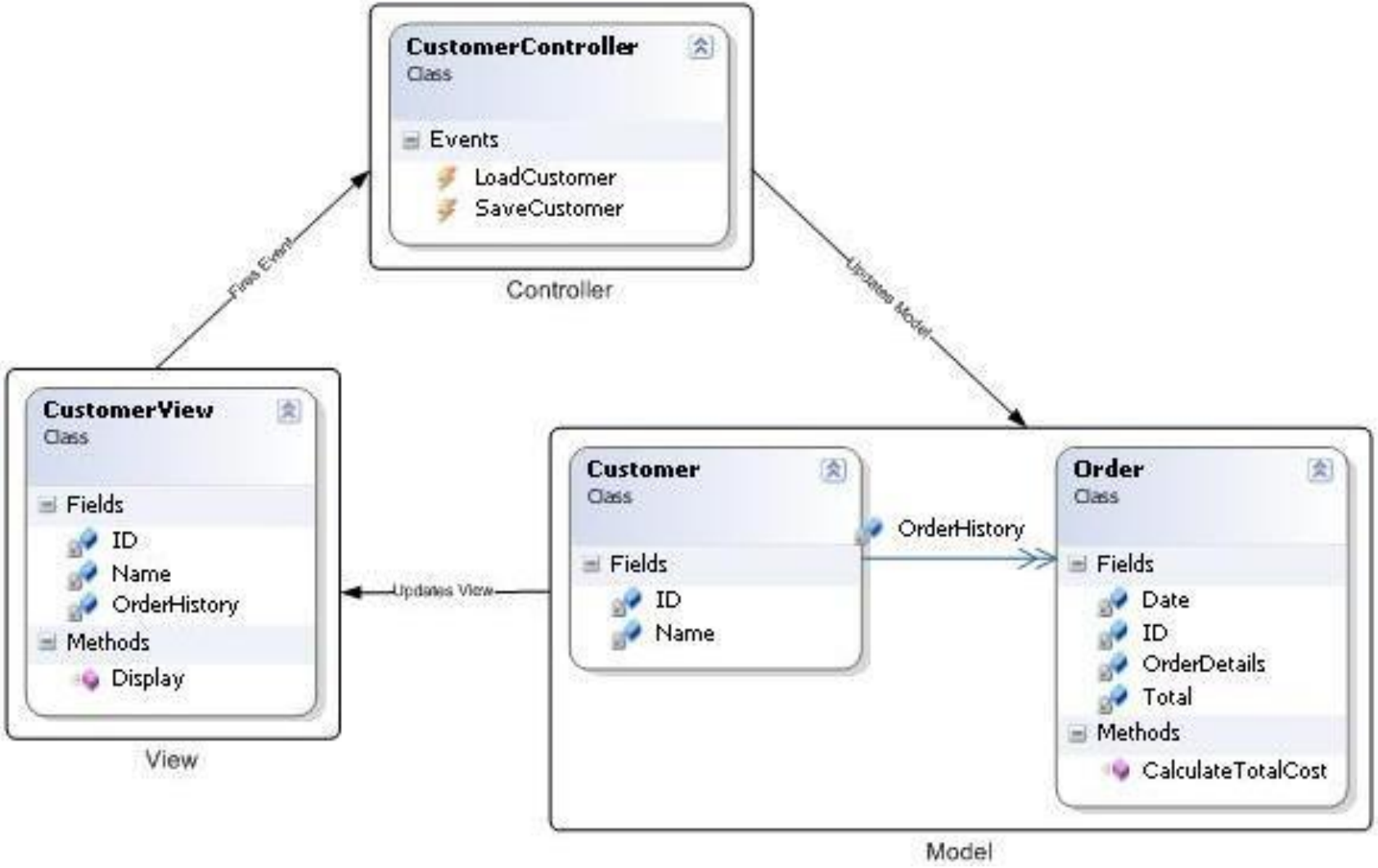
TUTORIAL DE MVC

Curso de ASP.NET MVC 4 C#(Csharp)



https://www.youtube.com/watch?v=LxpsTYFanx4

EJEMPLO - CLIENTES



3) PATRÓN – SENSE-COMPUTE-CONTROL

- Idea básica
  - ✓ Una computadora es embebida en alguna aplicación
  - ✓ Los sensores de distintos dispositivos están conectados a la computadora
  - ✓ Los sensores son interrogados para determinar su valor
  - ✓ La computadora también tiene asociados Actuators (dispositivos)
  - ✓ La computadora envía señales a los dispositivos a través de sus Actuators, logrando así controlar el sistema.

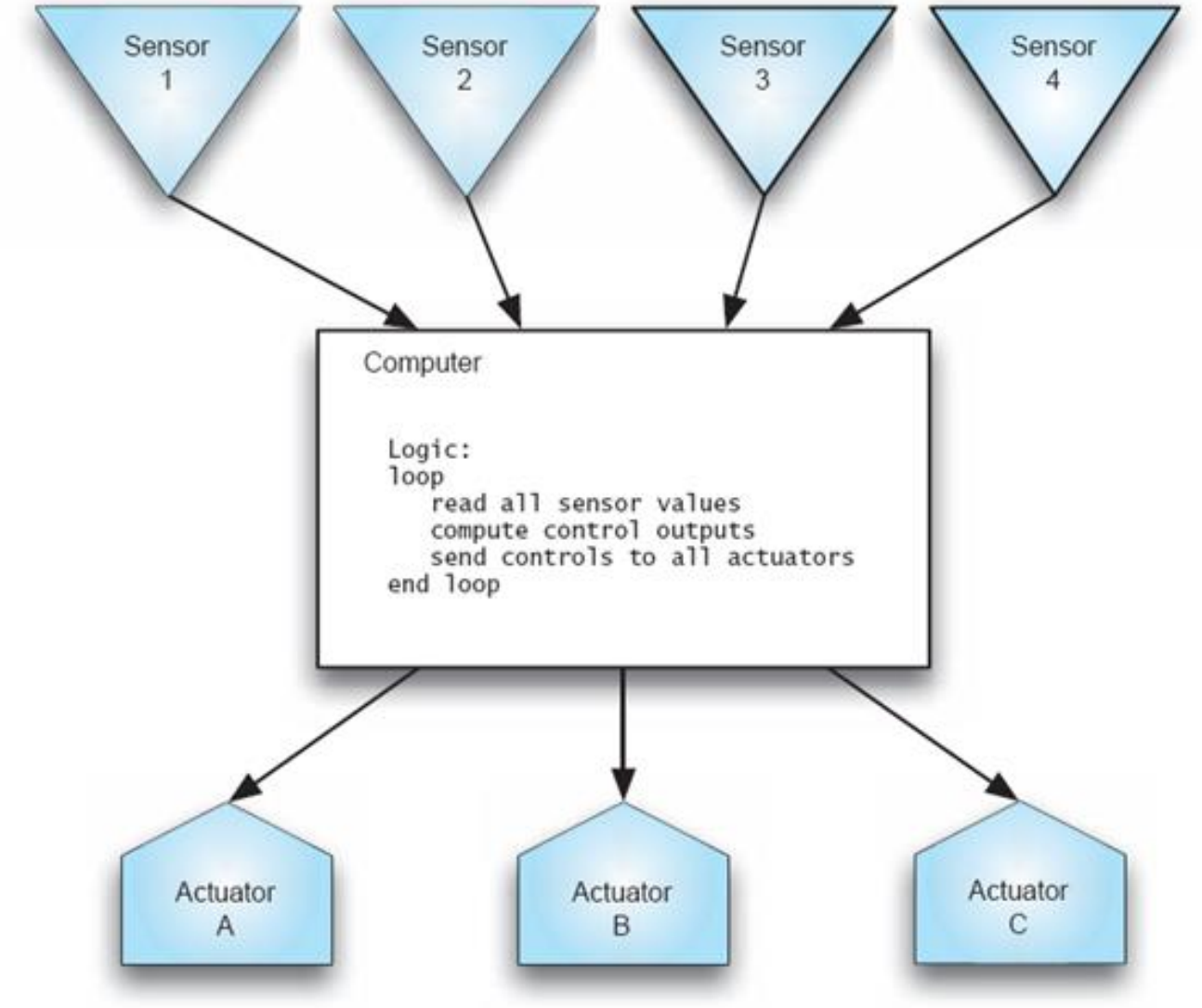
- Dónde se usa? Típicamente usado en aplicaciones de control embebidas
  - ✓ Simples electrodomésticos
  - ✓ Sistemas sofisticados para la industria automovilística
  - ✓ Control robótico.

3) PATRÓN – SENSE-COMPUTE-CONTROL - FUNCIONAMIENTO

- Cicla a través de los siguientes pasos:
  - ✓ Leer los valores de los sensores
  - ✓ Ejecutar un conjunto de leyes o funciones de control
  - ✓ Enviar la salida a los Actuators

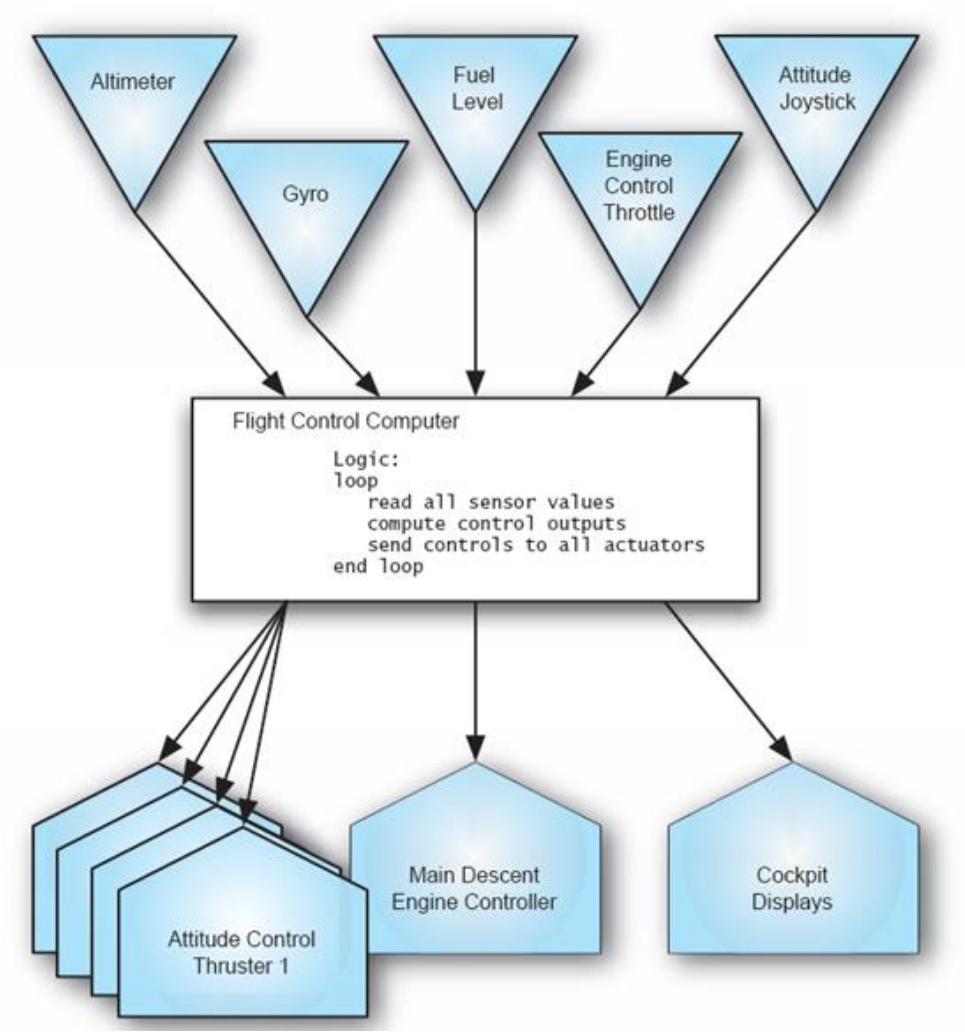
- Típicamente un ciclo está relacionado a un reloj
  - ✓ Frecuencia del reloj: Tasa máxima en que los valores de los sensores pueden cambiar, o en la sensibilidad con la que los Actuators pueden recibir actualizaciones.

3) PATRÓN – SENSE-COMPUTE-CONTROL - FUNCIONAMIENTO



3) PATRÓN – SENSE-COMPUTE-CONTROL – LUNAR LANDER

Ejemplo de control de vuelo



4) BROKER

CONTEXTO

- ✓ Algunos sistemas se construyen a partir de una colección de servicios distribuidos en múltiples servidores.
- ✓ Se requiere ocuparse de definir cómo los servicios van a interactuar (conectarse e intercambiar información) y manejar la disponibilidad de los servicios

PROBLEMA

- ✓ Cómo estructurar el software distribuido y que los usuarios de los servicios no necesitan conocer la naturaleza y ubicación de los proveedores de los servicios y hacer posible que cambien dinámicamente.

SOLUCIÓN

- El patrón broker separa a los usuarios de servicios (clientes) de los proveedores de servicios (servidores)

4) BROKER – OBJETIVOS Y FUNCIONAMIENTO

OBJETIVOS

- ✓ Separar a los usuarios del servicio (clientes) de los proveedores del servicio (servidores) insertando un intermediario, llamado broker

FUNCIONAMIENTO

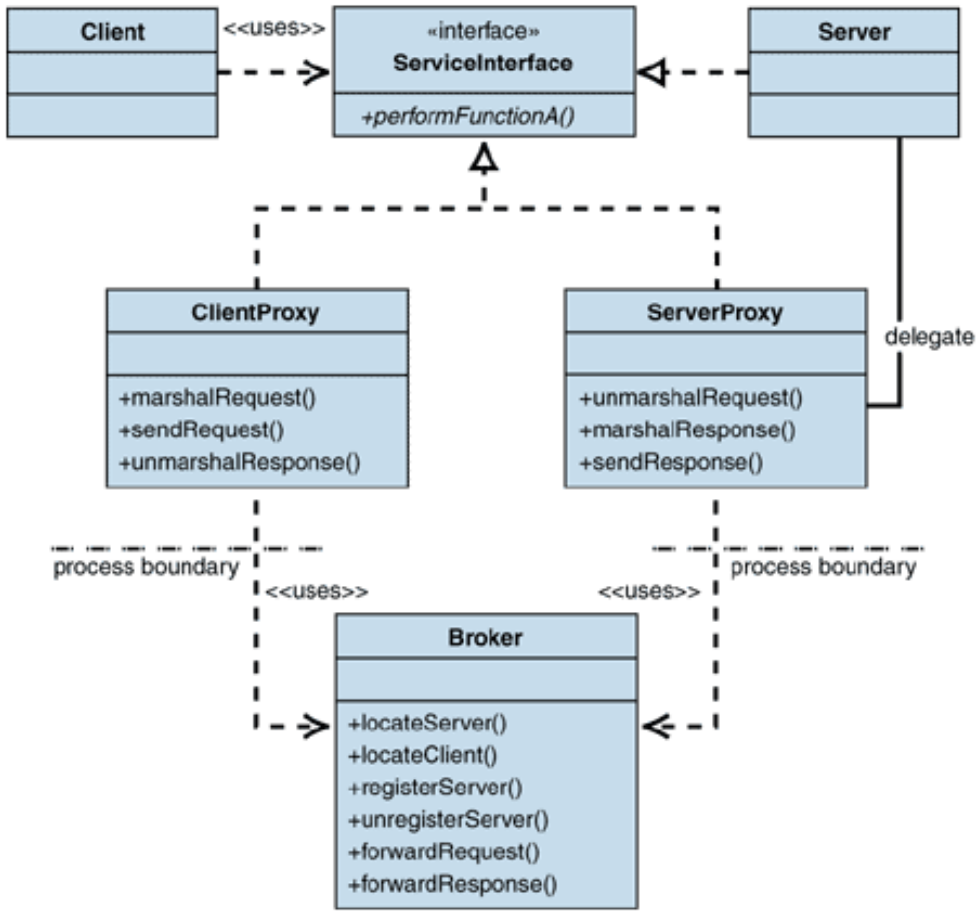
- ✓ Cuando el cliente requiere de un servicio, envía la consulta al broker
- ✓ El bróker reenvía el requerimiento del cliente a un servidor para que lo procese. El server envía el resultado al broker
- ✓ El broker retorna el resultado al cliente

4) BROKER – BENEFICIOS

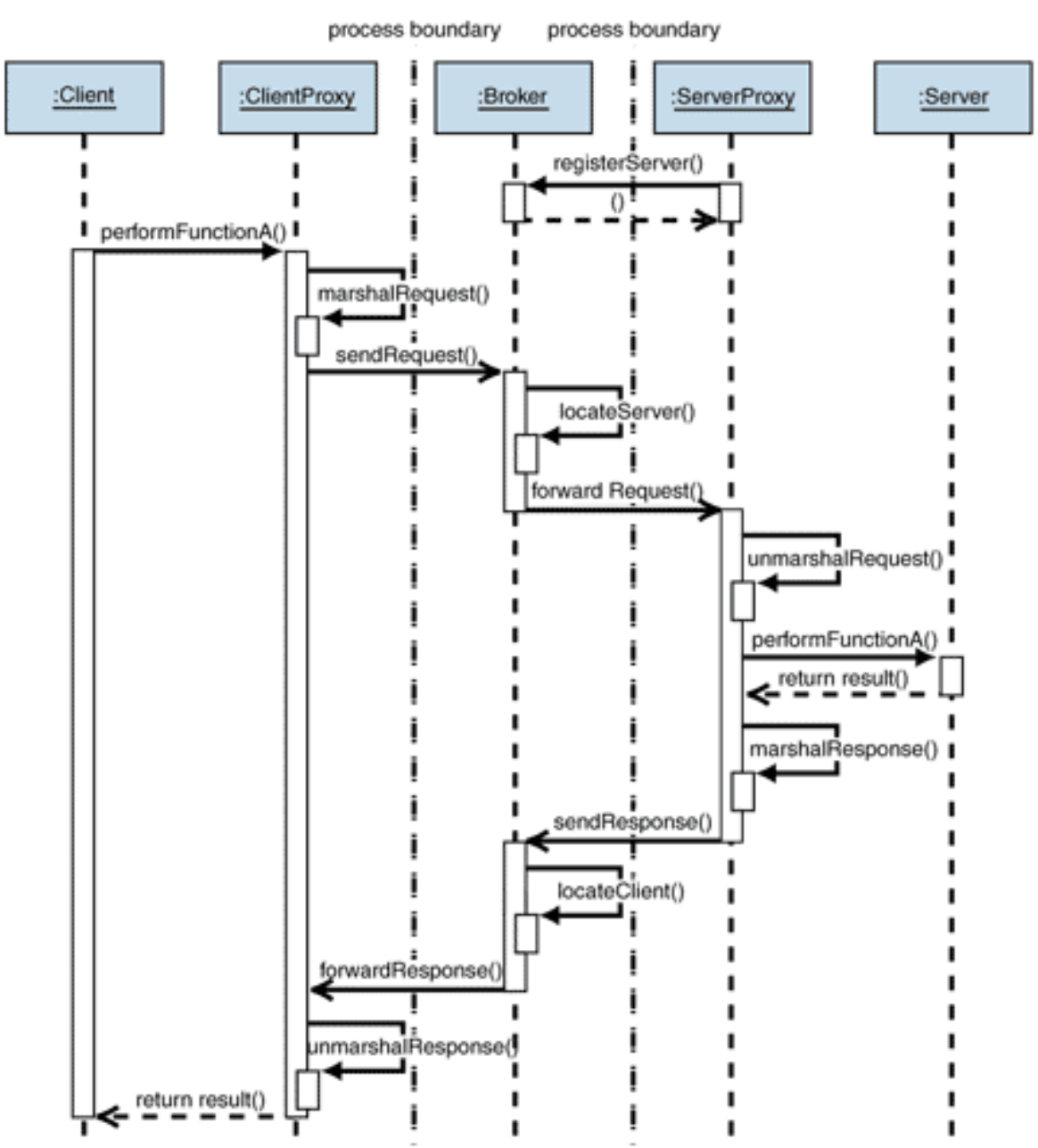
- Los clientes no tienen que conocer ni la identidad, ni la dirección, ni las particularidades de comunicación de los servidores
- Si un servidor se deja de estar disponible, el broker podría re-direccionar las peticiones a otro servidor que lo reemplace (disponibilidad)
- Si un servidor es reemplazado por otro, solamente el broker deberá conocer sobre dicho cambio (modificabilidad)

4) BROKER – COMPONENTES

- Client - consumidor de servicio
- Server -proveedor de servicio
- Broker - intermediario que localiza un server apropiado para cumplir el requerimiento de un cliente, reenvía el requerimiento al server y retorna el resultado al cliente
- Client-side proxy - un intermediario que maneja la comunicación real entre el cliente y el broker
- Server-side proxy - un intermediario que maneja la comunicación real entre el broker y el server



4) BROKER – FUNCIONAMIENTO



4) BROKER – DEBILIDADES

- agrega una capa de redirección y, por lo tanto, latencia entre clientes y servidores.
- dicha capa podría transformarse en un cuello de botella
- el broker podría ser un único punto de falla
- puede ser un objetivo de ataques de seguridad
- puede ser difícil de testear

4) BROKER – PRINCIPALES USOS

- CORBA
- Plataformas para proveedores de servicios distribuidos (EJBs, .NET)
- Service-Oriented Architectures (SOA) depende crucialmente de brokers, en la forma de Enterprise Service Bus (ESB)

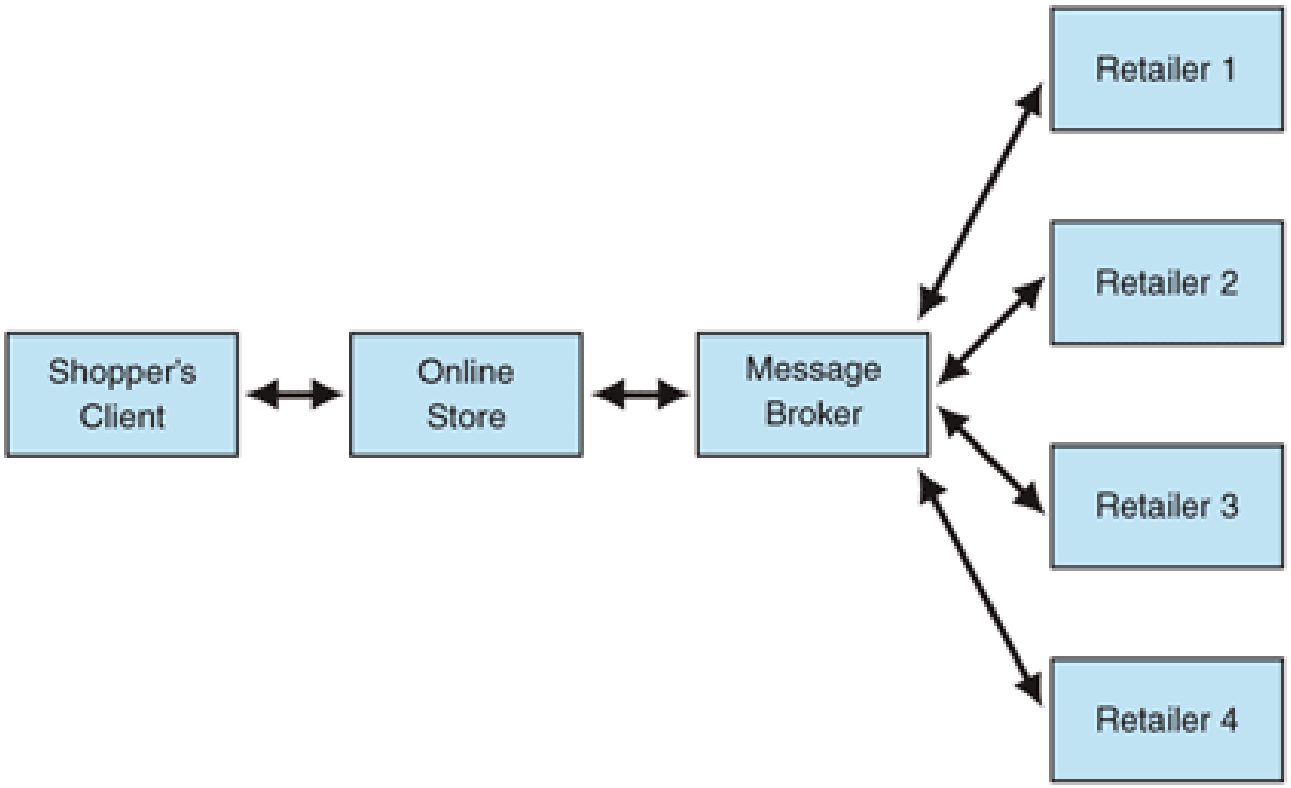
PATRÓN BROKER – ANÁLISIS 1

DESCRIPCIÓN	El patrón define un componente bróker en tiempo de ejecución que oficia de mediador de la comunicación entre clientes y servidores.
COMPONENTES	*Cliente,* que solicita un servicio *Servidor*, proveedor de servicios *Broker,* intermediario responsable de localizar un servidor apropiado para cumplir la petición del cliente, reenvía la solicitud al servidor, y devuelve los resultados al cliente *proxy del lado del cliente,* intermediario que resuelve la comunicación real con el bróker, incluye clasificar, enviar y desclasificar mensajes *proxy del lado del servidor,* intermediario que resuelve la comunicación real del servidor con el bróker, incluye clasificar, enviar y desclasificar mensajes

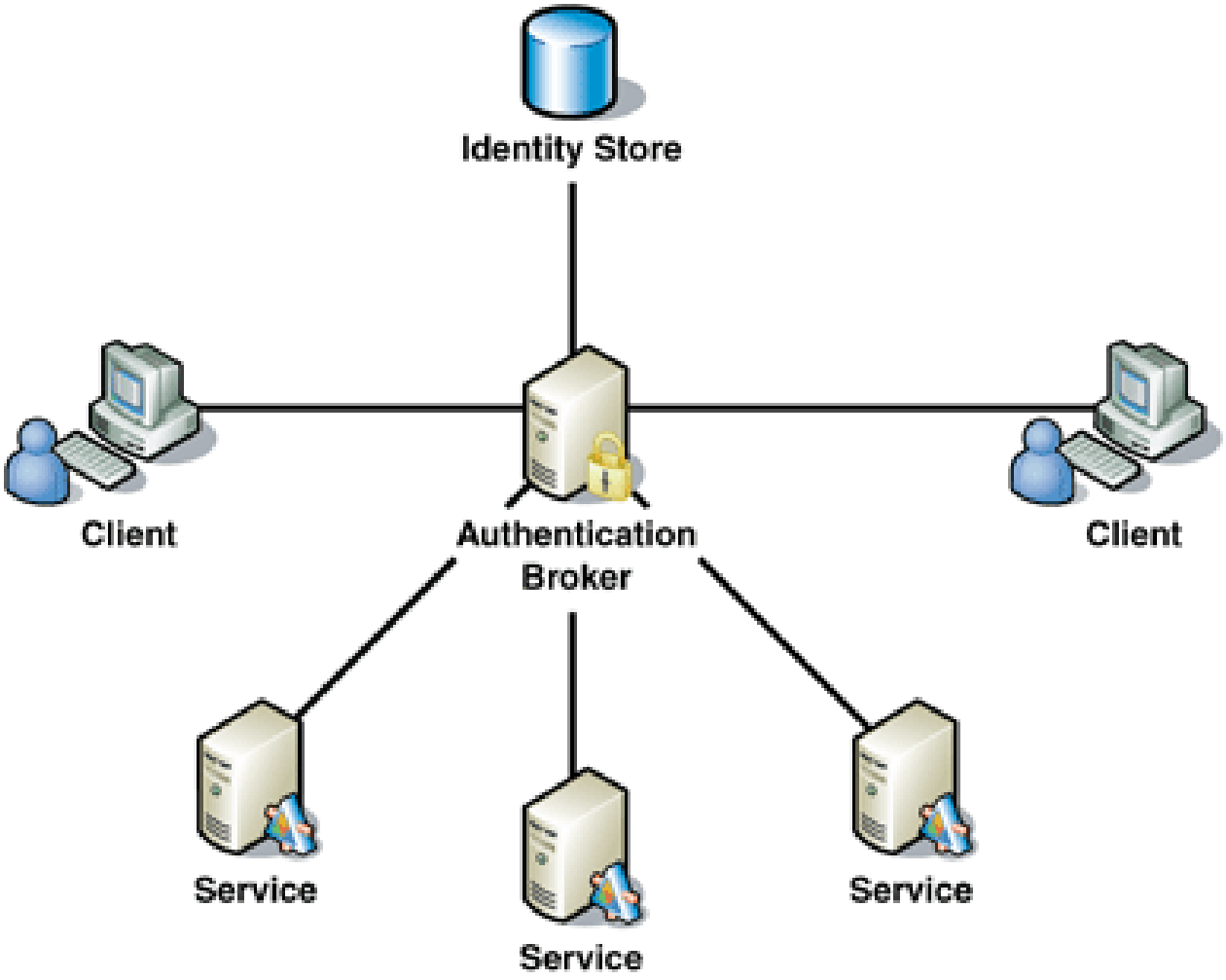
PATRÓN BROKER – ANÁLISIS 2

RELACIONES	La relación *adjuntar* asocia clientes (opcionalmente proxy-cliente) y servidores (opcionalmente proxy-server) con brokers
RESTRICCIONES	El cliente sólo se puede asociar con un broker (potencialmente via proxy-cliente). El servidor solo se puede asociar con un broker (potencialmente via proxy-server)
DEBILIDADES	• El broker agrega un nivel de indirección (aumenta la latencia y puede resultar en el cuello de botella) • El broker resulta el punto de falla. • Puede ser un objetivo de ataques de seguridad • Puede ser difícil de testear

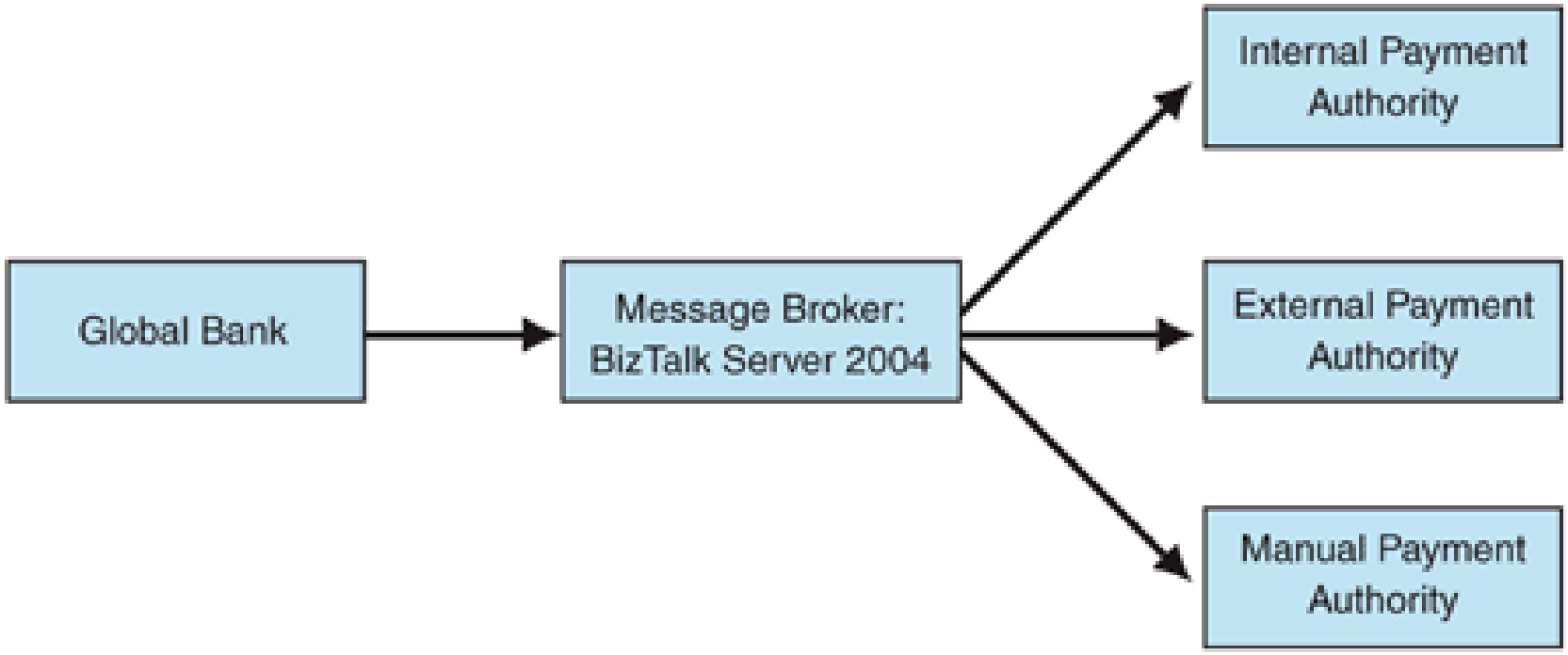
EJEMPLO – COMPRAS ONLINE



EJEMPLO – AUTENTICACIÓN



EJEMPLO – BANCO



Elsa Estevez
ece@cs.uns.edu.ar