



Arquitectura Web



Introducción

- | Concepto de Arquitectura en Desarrollo Software
 - | Concepción desde RUP
 - | Arquitectura física
 - | Distribución de nodos en la red
 - | Mapeo componente software – nodo computacional
- | Concepto de Arquitectura software Moderno
 - | Patrones de diseño de arquitectura
 - | Separación de responsabilidades
 - | No existe forma de representar arquitectura software con las herramientas actuales (RUP-UML)

Aplicaciones Web con Java

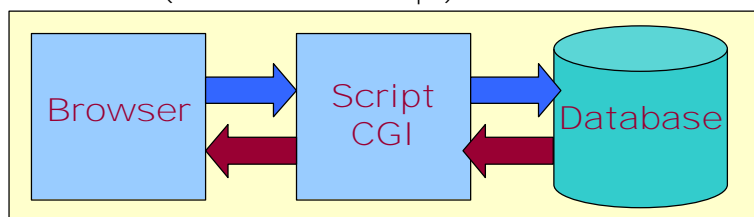
- i Fuerte apuesta por parte del sector privado:
 - | *Sun Microsystems*. Extensiones J2EE
 - | *BEA Systems* con [Weblogic](#)
 - | *IBM* con [WebSphere](#)
 - | *Netscape (y Sun)* con *iPlanet*
 - | *Orión – Oc4J* *Oracle 9IAS*
- i Fuerte apuesta del mundo opensource!
 - | www.apache.org Desarrollo del servidor web *apache*, el más difundido del mundo.
 - | Jakarta.apache.org Conjunto de frameworks y clases de utilidad como apoyo al desarrollo de aplicaciones basadas en java/J2EE.
 - | www.jboss.org Desarrollo del contenedor de EJBs *Jboss*. Gratuito y muy efectivo.

Evolución de Modelos Arquitectónicos

- i Modelo 1  Servlets/JSPs
- i Modelo 1.5 MVC Model
- i Modelo 2
- i Modelo 2X Multicanalidad

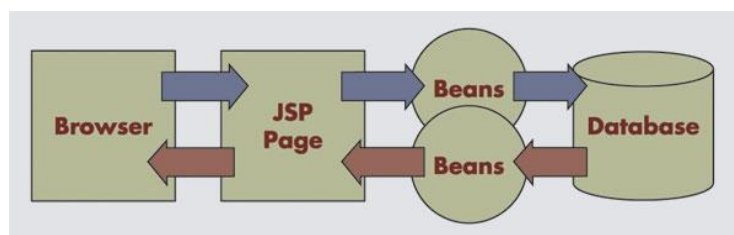
Modelo de Arquitectura 1 Aplicaciones CGI

- | Las más primitivas
- | Aplicaciones Web CGI
- | Presentación, negocio y persistencia mezclados
- | El estado se almacena en el cliente y cada petición supone una ejecución completa independiente de estado (Transaction Script)



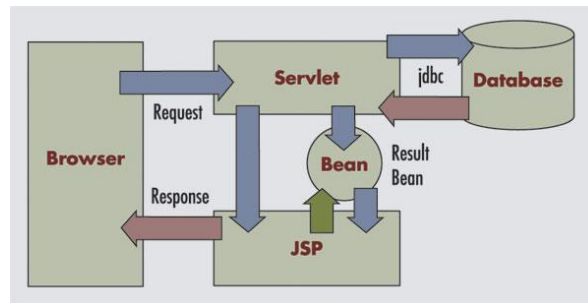
Modelo de Arquitectura 1.5 JSP y Servlets

- | Separación de responsabilidades:
 - | JSPs llevan la lógica de presentación (navegabilidad, visualización, etc.)
 - | Beans incrustados asumen las responsabilidades de negocio y datos



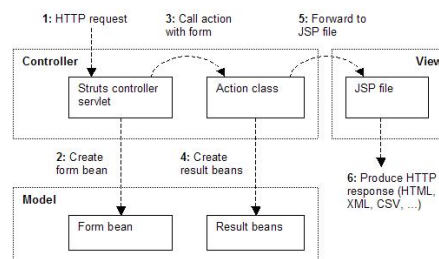
Modelo de Arquitectura 2 MVC

- Evolución del modelo 1.5
- Incorporación del patrón de diseño MVC.
 - Controlador: Navegación
 - Negocio y Datos: Beans
 - Presentación: JSPs



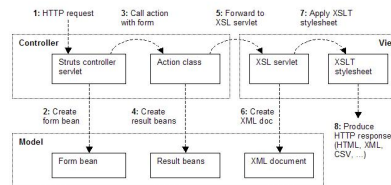
Modelo de Arquitectura 2 MVC con Struts

- Struts es la implementación del MVC que aporta Jakarta para aplicaciones web java.
- <http://jakarta.apache.org/struts>



Modelo de Arquitectura 2X Aplicaciones Multicanal

- i Evolución del modelo 2 para construir aplicaciones multicanal.
- i Implementación de referencia STXX (extiende Struts)
- i <http://stxx.sourceforge.net/>
- i Soluciones basadas en XML y XSLTs.



Aspectos Generales en Arquitectura WEB

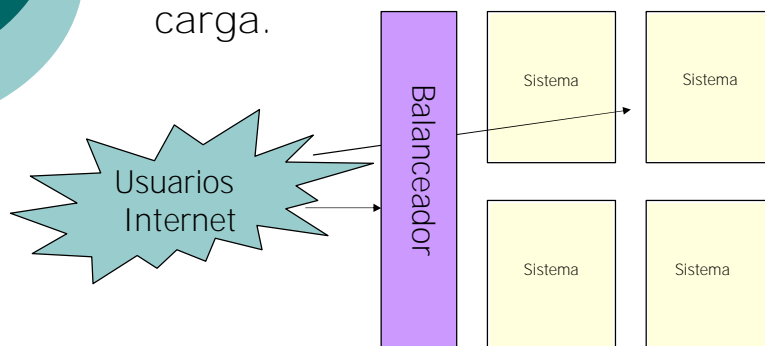
- i Escalabilidad
- i Separación de responsabilidades
- i Portabilidad
- i Componentización de los servicios de infraestructura
- i Gestión de la sesión del usuario, cacheado de entidades
- i Aplicación de patrones de diseño

Escalabilidad ¿Importancia?

- i Característica principal apps WEB:
 - | Posible incremento vertiginoso del número de usuarios
- i Es importante:
 - | El correcto dimensionamiento de la aplicación
 - | La adaptabilidad del sistema ante el incremento de demanda.
- i Varias opciones:
 - | Escalabilidad Horizontal
 - | Escalabilidad Vertical
 - | Cluster de servidores

Escalabilidad Horizontal

- i Clonamos el sistema y balanceamos la carga.





Escalabilidad Horizontal. Balanceador HW

- i Distribuye por algoritmos predeterminados (Round Robin, LRU, etc.) las peticiones HTTP entre los distintos clones del sistema
- i La selección del clon es por tanto aleatoria
- i Problema: No garantiza que diferentes peticiones de un mismo usuario sean servidas por el mismo clon del sistema
-> No hay mantenimiento de la sesión del usuario en servidor -> Condiciona el Diseño!
- i La sesión la debe mantener el desarrollador por otros medios:
 - | Cookies
 - | En base de datos
- i Al ser un proceso HW, es MUY rápido.



Escalabilidad Horizontal. Balanceador SW

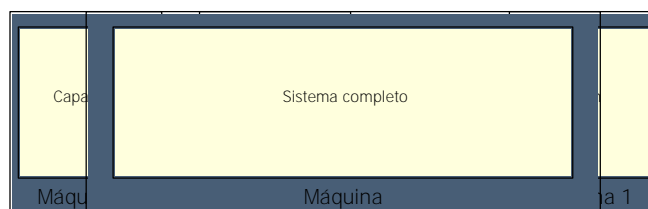
- i Examinan el paquete a nivel del protocolo HTTP para garantizar el mantenimiento de la sesión de usuario.
- i Distintas peticiones del mismo usuario son servidas por el mismo clon del servidor.
- i Más lentos que los balanceadores HW
- i Normalmente son soluciones baratas.
- i Ej., módulo mod_jk de apache.

Escalabilidad Horizontal. Balanceador HW HTTP

- i Dispositivos HW que examinan la petición a nivel de paquete HTTP.
- i Término medio entre las dos anteriores.
- i **Garantizan el mantenimiento de sesión.**
- i Más rápidos que los SW pero menos que los HW.

Escalabilidad Vertical


- i La separación lógica entre capas se implementa de forma que permita la separación física de las mismas.
- i Es necesario un **Middleware** entre las capas para permitir la comunicación remota.





Escalabilidad Custers de Servidores

- i Habituales en los servidores de aplicaciones comerciales (Weblogic, WebSphere, iPlanet, etc.).
- i Dependiendo de cómo se aplique puede clasificarse como horizontal o vertical.
- i Distribuye y escala el sistema de modo transparente a usuario y administrador.
- i Garantiza que sea cual sea la máquina que sirva la petición http tendrá acceso a la sesión del usuario (Replicación de sesión)
- i La replicación de sesión es MUY costosa, produce bajo rendimiento del sistema.



Entonces... ¿Qué hacer con la sesión?

- i Primeras tendencias eran evitar apoyarse en la sesión (objeto *Session*): sólo había balanceadores hw.
- i Hoy en día, está aceptado y se fomenta su uso.
- i OJO! Es MUY delicado. El uso excesivo del objeto *session* puede acarrear problemas de rendimiento, puesto que los objetos en sesión no se liberan hasta que no caduque la misma.

Separación de Responsabilidades

- i Premisa base para la separación de capas
- i Distintas Responsabilidades no deben ser delegadas en la misma clase (*separación de incumbencias*)
- i Tendencia actual en aplicaciones WEB:
 - i Arquitectura *n-capas*
- i El modelo más básico es el de tres capas:
 - i Capa de presentación
 - i Capa de negocio
 - i Capa de persistencia
- i Independencia de capas

Separación de Responsabilidades – Evolución

- APLICACIONES MAINFRAME



Única capa física y lógica

Separación de Responsabilidades – Evolución

- APLICACIONES CLIENTE - SERVIDOR

CLIENTE

PRESENTACIÓN
Y NEGOCIO

SERVIDOR

NEGOCIO Y
ACCESO A DATOS

Separación Lógica y Física de la
interfaz de usuario

Separación de Responsabilidades – Evolución

- PRIMERAS APLICACIONES WEB

Arquitectura basada en Transaction
Scripts (CGIs, Modelo 1)

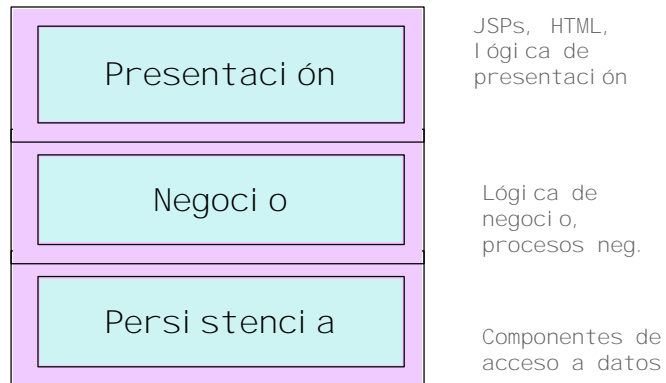
Presentación

Negocio +
acceso a datos

Interfaz WEB – HTML
+ lenguaje de script

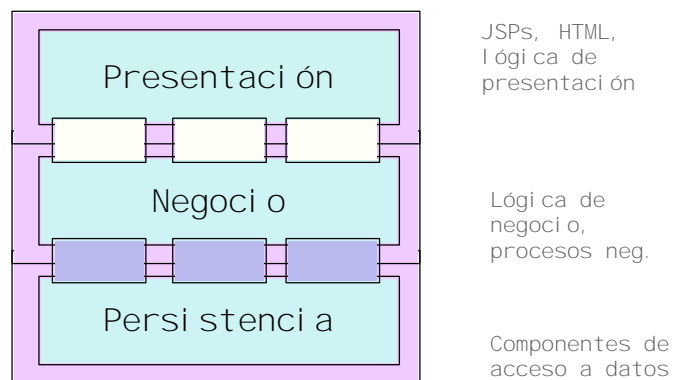
Separación de Responsabilidades – Evolución


- APLICACIONES 3 CAPAS



Separación de Responsabilidades – Evolución


- Modelo de Brown ncapas






Separación de Responsabilidades – Capa de presentación

- i Comprende las responsabilidades de lógica de presentación:
 - | Navegabilidad del sistema
 - | Validación de datos de entrada
 - | Formateo de los datos de salida
 - | Internacionalización
 - | Renderizado de presentación
 - | Etc.



Separación de Responsabilidades – Capa de negocio

- i Comprende las responsabilidades de lógica de negocio (o dominio) del sistema.
- i Resultado del análisis funcional:
 - | Conjunto de reglas de negocio que abstraen el mundo real.
- i La capa de negocio ha de ser independiente de la capa de presentación y viceversa (en la medida de lo posible).



Separación de Responsabilidades – Capa de persistencia

- i Comprende las responsabilidades de lógica de persistencia de las entidades que maneja el sistema en desarrollo.
 - | Inserción
 - | Eliminación
 - | Actualizaciones
 - | Búsquedas
 - | Etc.
- i No tiene porqué tratarse necesariamente de una base de datos relacional.



Portabilidad

- i Una aplicación web debe poder adaptarse a las distintas arquitecturas físicas posibles en el despliegue.
- i Las tareas de adaptación a un nuevo entorno deben limitarse al ámbito de la configuración, no del desarrollo.
- i *Supuesto de ejemplo: Cliente reacio a las tecnologías de componentes J2EE (EJBs) por costes, rendimiento o simplemente, moda.*



Componentización de los servicios de infraestructura

- | ¿Servicio de infraestructura?: Componentes independientes del dominio.
- | Rompen aparentemente la separación vertical de capas.
- | Dan lugar a la capa de infraestructura.
- | Ej.:
 - | Servicio de Log
 - | Pool JDBC
 - | Sistema de configuración
 - | Gestor de permisos de acceso
 - | Etc.



Gestión de la sesión del usuario

- | Aspecto muy delicado del sistema
- | Cacheado de entidades en
 - | Sesión de usuario
 - | Contexto de la aplicación
- | Caducidad de la información
- | Refresco de datos
- | Rendimiento del sistema. Consumo de recursos del sistema.



Aplicación de patrones de Diseño

- i Definición de patrón de diseño
- i GOF 94 *Design Patterns*
- i Además de una solución válida para problemas habituales, son un **medio de entendimiento** que facilita la comunicación entre analista y desarrollador.
- i Aceleran el desarrollo de Software
- i Facilitan el mantenimiento
- i En proceso de integración en las herramientas CASE (Rose, Together, etc.).