# Evolución de Arquitecturas y Frameworks para sistemas J2EEE

**María Consuelo Franky**

**CincoSOFT Ltda.**
**ConsueloFranky@cincosoft.com**
**http://www.cincosoft.com**


Septiembre 27 a Octubre 01 de 2005
Bogotá, Colombia

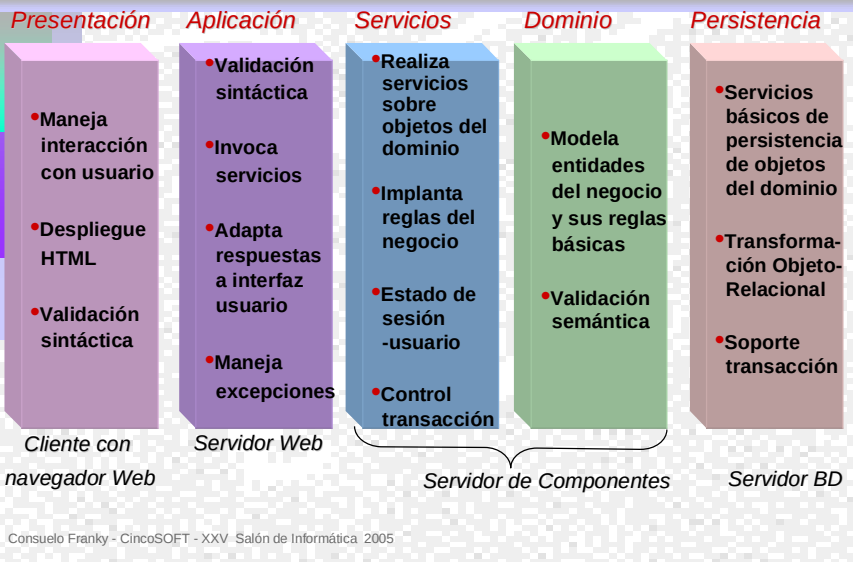
## Temario

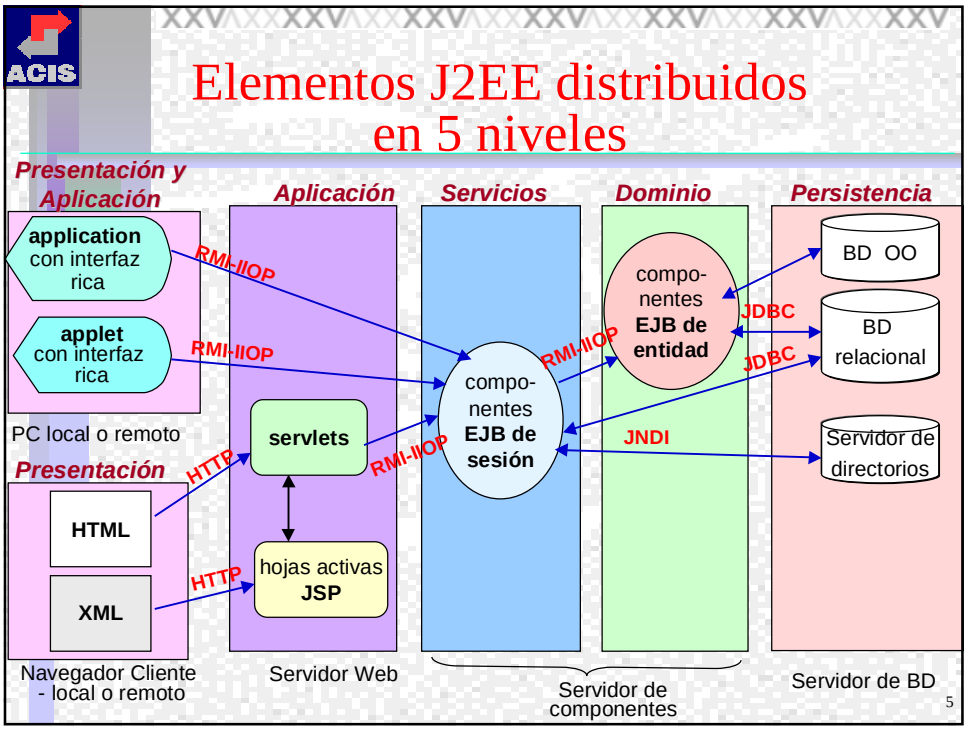
- Elementos de un sistema multi-nivel J2EE
- Organización de un sistema multi-nivel bajo el metapatrón MVC
- Frameworks para construir sistemas multi-nivel
- Evolución de los frameworks para componentes Web
- Evolución de los frameworks para componentes de negocio
- Conclusiones y Bibliografía

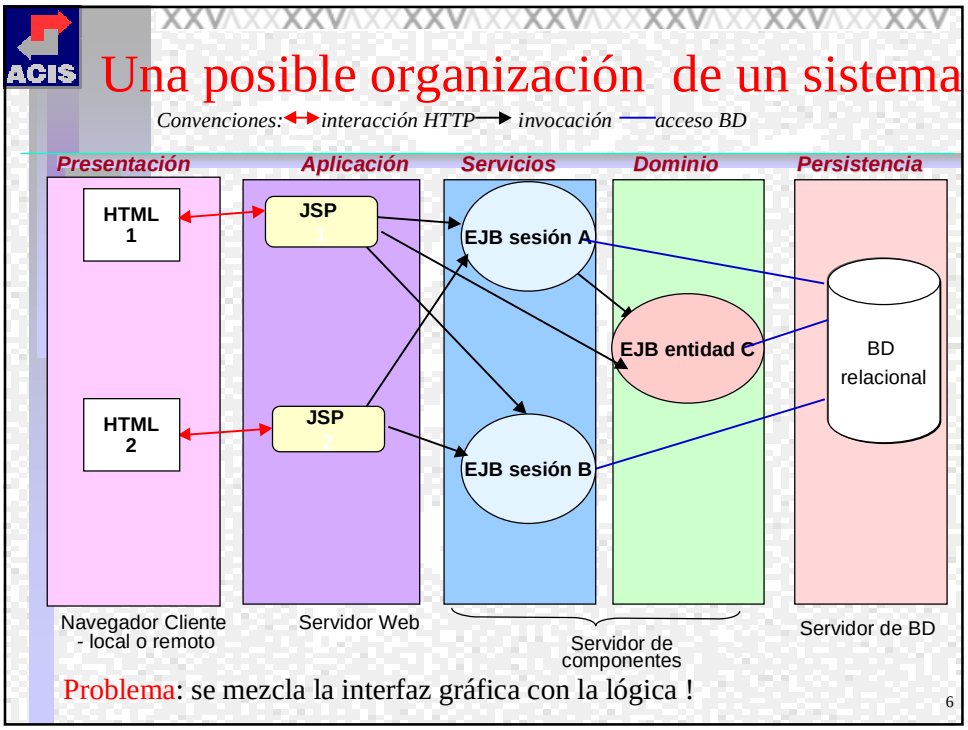
# 1.

# Elementos de un sistema multi-nivel J2EE

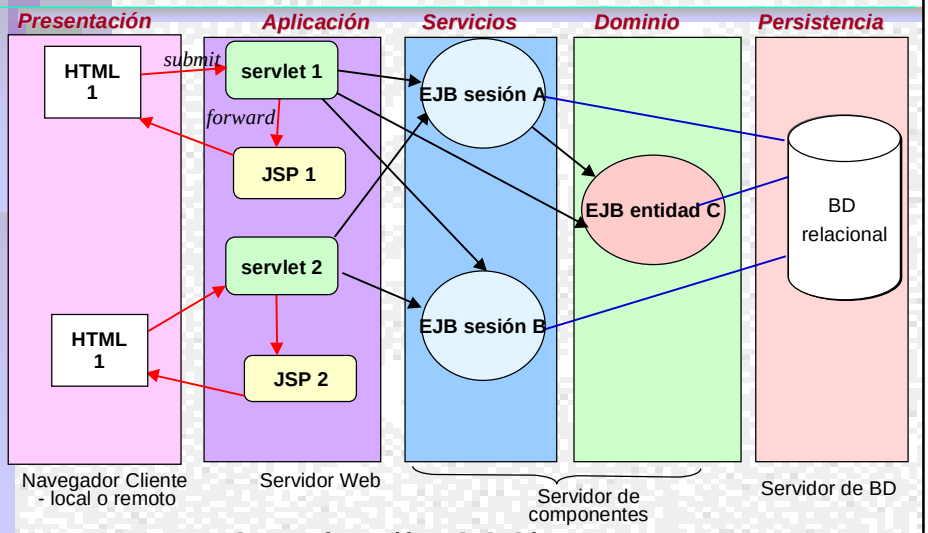
## Niveles de una arquitectura multi-nivel




ACIS
Elementos J2EE distribuidos en 5 niveles
Presentación y Aplicación
Aplicación
Servicios
Dominio
Persistencia
application con interfaz rica
applet con interfaz rica
PC local o remoto
RMI-IIOP
RMI-IIOP
servlets
HTTP
Presentación
HTML
XML
HTTP
hojas activas JSP
compo-nentes EJB de sesión
RMI-IIOP
compo-nentes EJB de entidad
JDBC
JDBC
JNDI
BD OO
BD relacional
Servidor de directorios
Navegador Cliente - local o remoto
Servidor Web
Servidor de componentes
Servidor de BD
5


ACIS
Una posible organización de un sistema
Convenciones: interacción HTTP invocación acceso BD
Presentación
Aplicación
Servicios
Dominio
Persistencia
HTML 1
HTML 2
JSP 1
JSP 2
EJB sesión A
EJB sesión B
EJB entidad C
BD relacional
Navegador Cliente - local o remoto
Servidor Web
Servidor de componentes
Servidor de BD
Problema: se mezcla la interfaz gráfica con la lógica !
6

# Otra posible organización de un sistema




Mejora: se separa la interfaz gráfica de la lógica

7

# Problemas => ineficiencia, mantenimiento difícil

- Explosión de servlets y JSP
- Baja reutilización de elementos del nivel de Aplicación
- Navegación entre pantallazos difícil de mantener
- Referencias remotas pesadas hacia componentes EJB
- Instanciación repetida de los mismos componentes EJB

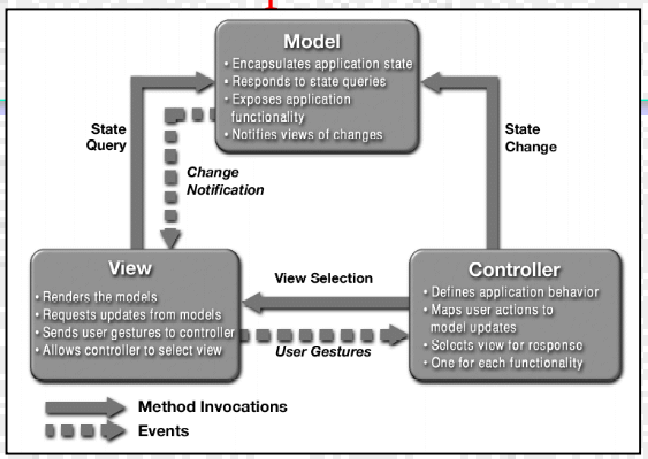
# 2.

# Organización de un sistema multi-nivel bajo el metapatrón MVC

## Objetivos del diseño

- Reutilización del código y del diseño
- Descomposición funcional
- Desarrollo paralelo de subsistemas que requieren distintas habilidades
- Extensibilidad
- Modularidad
- Seguridad
- Presentación estándar
- Minimización de tráfico de red
- Múltiples interfaces-usuario

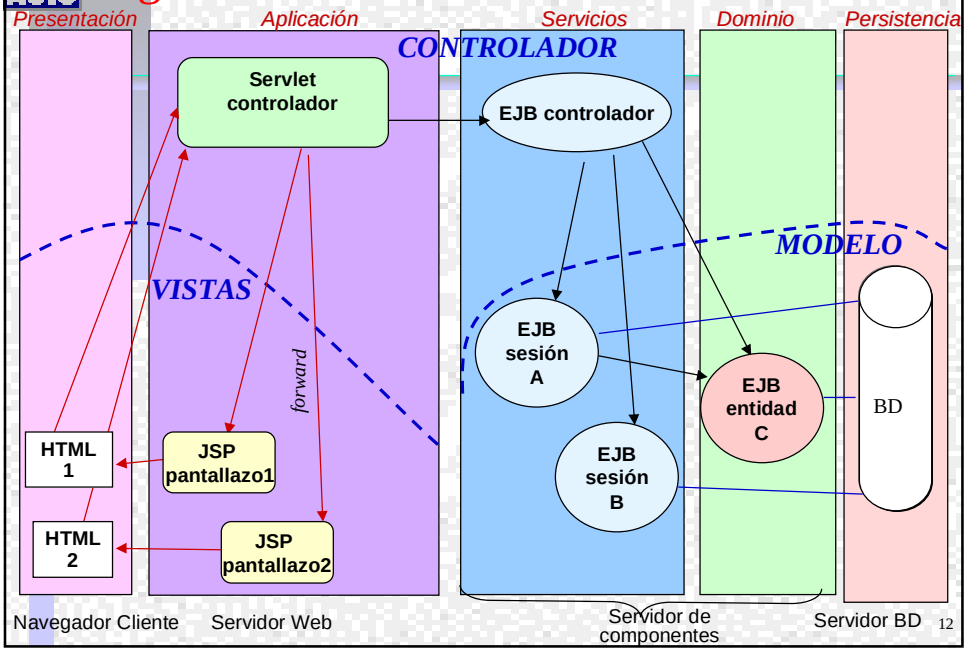
# El metapatrón MVC




- **View (Vistas)**: maneja presentación de los datos del sistema al usuario
- **Model (Modelo):** maneja reglas del negocio y estructura de los datos
- **Controller (Controlador):** transforma pedidos del usuario en operaciones sobre los objetos del modelo y selecciona vista para mostrar resultados al usuario (comportamiento del sistema)

11

# Organización MVC de un sistema multi-nivel



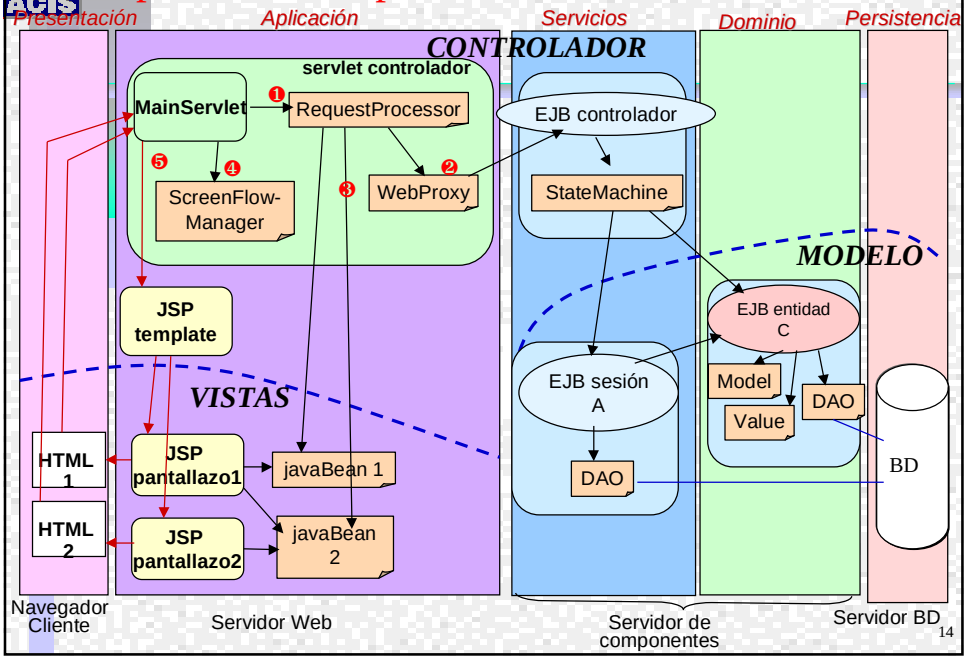

12

# Problemas resueltos
# => eficiencia, mantenimiento, flexibilidad

- Factorización de servlets en un servlet Controlador:
  - centraliza autenticación, manejo de timeout, despacho de pedidos, selección de pantallazo, etc.
- Información centralizada sobre navegación entre pantallazos
- Alta posibilidad de reutilización de elementos del nivel de Aplicación
- El servlet Controlador mantiene solo una referencia remota hacia el EJB controlador:
  - las demás referencias hacia componentes EJB son locales.
- Reutilización de las instancias EJB



# Implantación de patrones en un sistema multi-nivel



## Problemas en la implantación de patrones

- Larga curva de aprendizaje
- Se requiere personal muy capacitado
- Elevados costos y tiempos de desarrollo de proyectos
- Alta probabilidad de:
  - implantar los patrones de forma errada
  - generar bugs en la implantación
- Cada nuevo proyecto es volver a comenzar la implantación de patrones ...



# 3.

# Frameworks para construir sistemas multi-nivel

# Qué es un framework ?

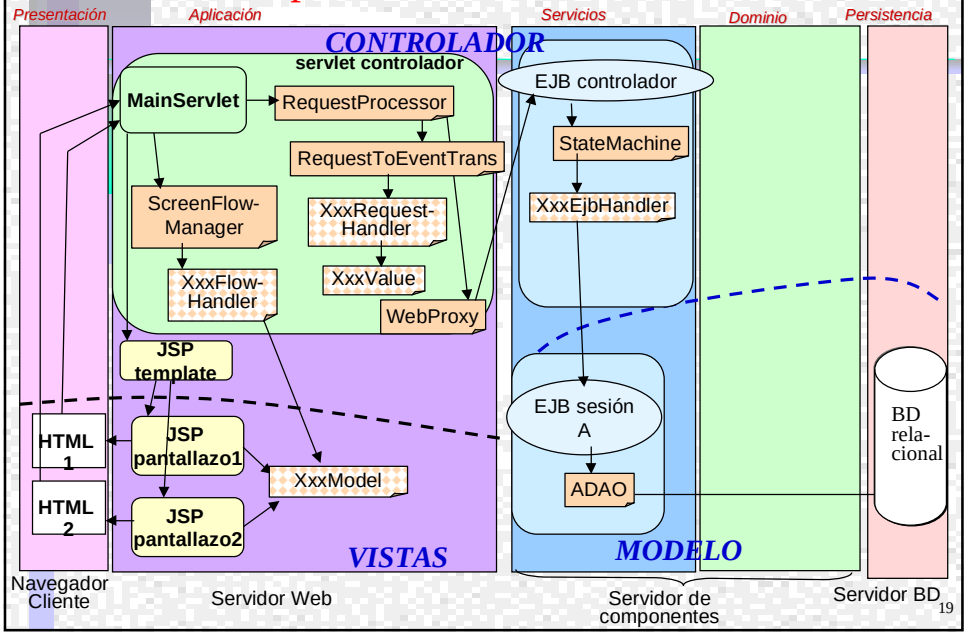
- Un framework es un conjunto de servicios y componentes reutilizables organizados en una estructura extensible, que busca simplificar el desarrollo de aplicaciones
- Los frameworks
  - reducen el tiempo de desarrollo de los proyectos
  - reducen el tiempo de entrenamiento de los desarrolladores
  - reducen la curva de aprendizaje para los desarrolladores
- Un buen framework asegura sistemas eficientes y escalables
  - mediante la implantación de patrones



# Frameworks particulares construidos en las empresas : el caso de CincoSOFT

- Un framework propio de la empresa debería:
  - materializar estandarizaciones, buenas prácticas y reutilizaciones en todos los niveles
  - producir software homogéneo y bien documentado
  - cubrir todos los elementos del proceso (fuentes, pantallazos, descriptores, ant)
- Características del framework CincoSOFT:
  - se apoya en una infraestructura básica que implementa el metapatron MVC
  - automatiza el desarrollo por casos de uso
  - la automatización se logra a través de plantillas **Velocity** (herramienta Apache) que representan las clases de un caso de uso.
  - está en permanente construcción (investigación permanente)

# Framework CincoSOFT: clases generadas para un caso de uso Xxx




# Propiedades del caso de uso Users

```
# name of the project:
project = acme

# author of sources:
author = CincoSOFT

# absolute root path of files to be generated
root = f:/Mcf/ProyACME

# name of the module (ear) containing this usecase:
module = acme

# name of the usecase:
usecase = users

# list of screens of the usecase:
listScreen = UsersProblemMVC,UsersMainMVC,UsersDetailsMVC

# list of MVC actions requested to the server by the usecase:
listAction = LIST,GET_USER,SAVE_USER

# does the user interface uses html tabs? write true or false
hasHTMLtabs = false
. . . . .
```

# Plantilla Velocity de una clase del caso de uso: procesamiento de los pedidos del usuario

```
package com.${project}.${module}.control.web.handlers;
import com.${project}.${module}.${usecase}.modelvalue.${Usecase}Value;
/**
 * ${usecase} usecase: request handler in the Application layer
 * @author  $author
 */
public class ${Usecase}RequestHandler extends RequestHandlerSupport{
 /** principal method that process a user request */
  public ${Module}Event processRequest
    (HttpServletRequest request, ServletContext context){
    #foreach ($action in $listAction)
    if(action.equals("${action}")) {
      return createEventFor${action}(request);
    #end
    return null;
  }
  ....
```



# Clase generada a partir de la plantilla: UsersRequestHandler

```
package com.acme.acme.control.web.handlers;

import com.acme.common.utilcomponents.UsersValue;
/**
 * users usecase: request handler in the Application layer
 * @author  CincoSOFT
 */
public class UsersRequestHandler
                          extends RequestHandlerSupport{
 /** principal method that process a user request */
  public AcmeEvent processRequest
    (HttpServletRequest request,ServletContext context){
    if(action.equals("LIST")) {
      return createEventForLIST(request);
    }
    if(action.equals("GET_USER")) {
      return createEventForGET_USER(request);
    }
    if(action.equals("SAVE_USER")) {
      return createEventForSAVE_USER(request);
    }
    return null;
  }
....
```

## Archivos generados o extendidos para el caso de uso

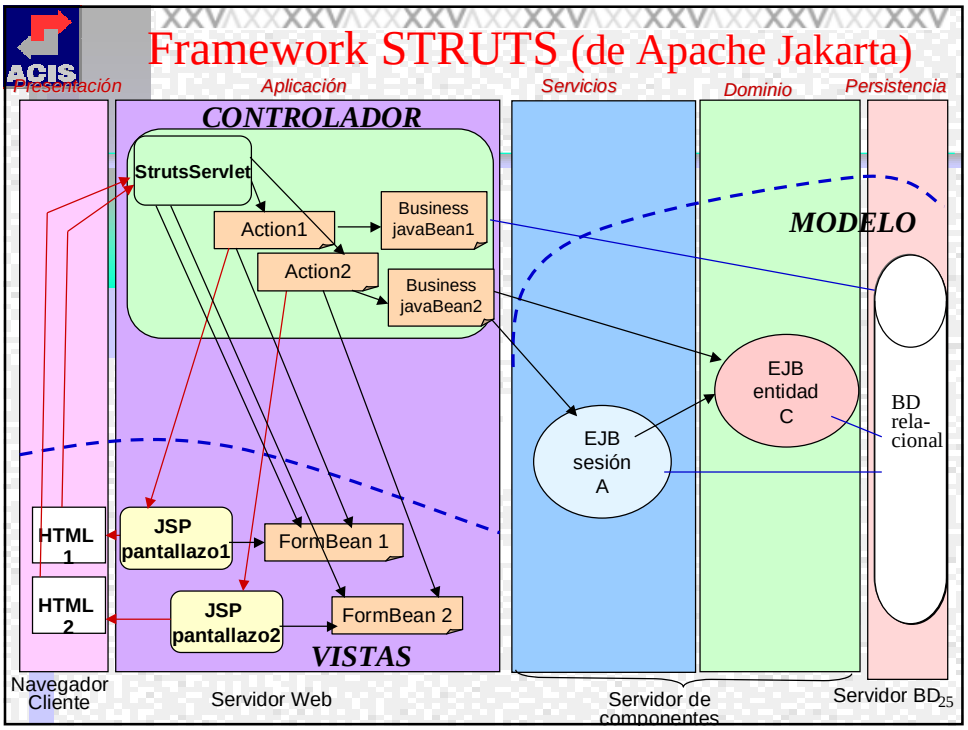
- Generación del contenido inicial de:
  - Clases básicas para el caso de uso: auxiliares del Controlador
  - Pantallazos con diversos elementos HTML según valor de propiedades:
    - con tarjetas o no
    - obtienen valores del Modelo a través de un javaBean
- Extensión de:
  - nuevos servicios en los componentes EJB
  - descriptores J2EE (web.xml, ejb-jar.xml, . . .)
  - archivos que describen navegación entre pantallazos
  - tareas ANT



# 4.

# Evolución de los frameworks para componentes Web

# Framework STRUTS (de Apache Jakarta)




# Facilidades de STRUTS:

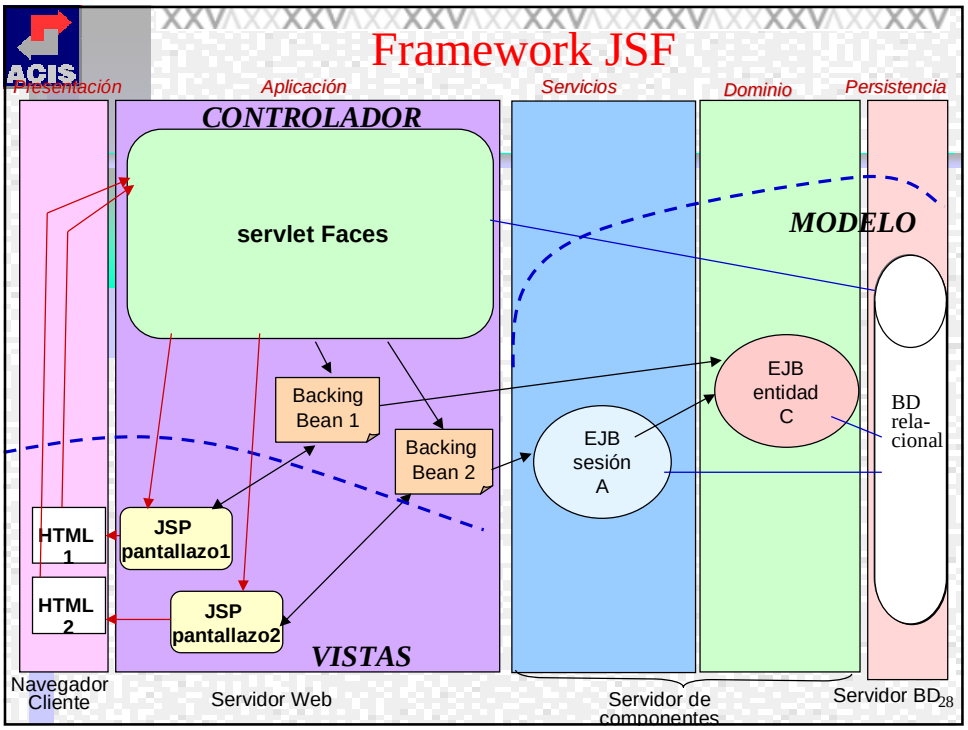
- Servlet Controlador
- Custom tags (etiquetas) para construir JSP
- Generación de clases FormBean que representan formas HTML para cada tipo de pedido
- facilidades de validación sintáctica
- Clase maestra Action para definir procesamiento de cada tipo de pedido

# JSF: Java Server Faces

- Estándar adoptado en J2EE para construir niveles de Presentación y Aplicación
  - Java Community Process JSR-127
  - JSF 1.2 será parte de J2EE 5.0
- Pantallazos (jsp) se contruyen con componentes gráficos estándares que reaccionan a eventos (UIComponent):
  - los UIComponent encapsulan elementos HTML
  - diversos tipos: UIInput, UIOutput, UICommand, …con validaciones estándares
  - Muestran y actualizan valores del Modelo contenidos en javaBeans ("backing beans")
  - los "backing beans" también procesan eventos asociados a los UIComponents (pueden invocar componentes de negocio)
  - archivos de configuración: navegación entre pantallazos, backing beans ...

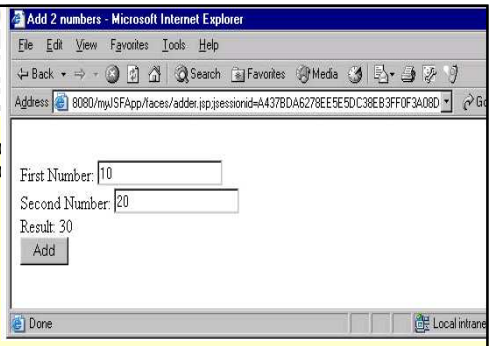
27

# Framework JSF




28

## Ejemplo de pantallazo en JSF:

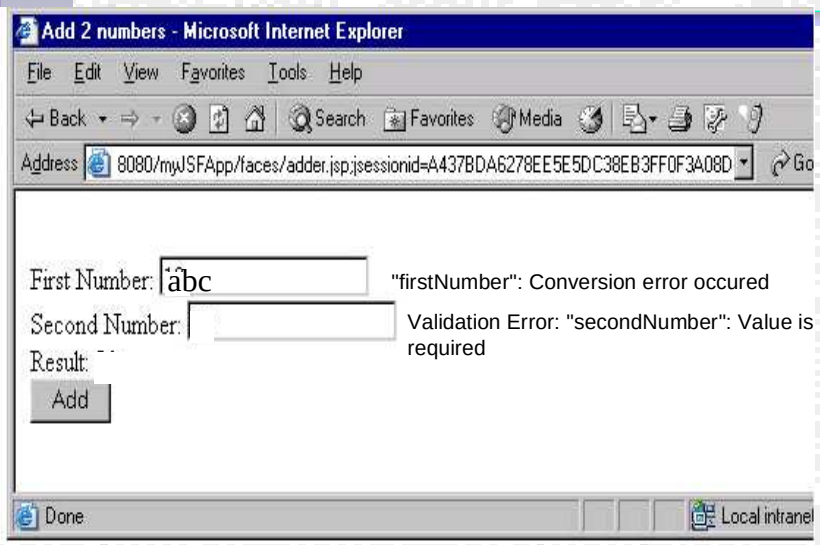



```
<%@ taglib uri="http://java.sun.
<%@ taglib uri="http://java.sun.
<html>
<head>
<title>Add 2 numbers</title>
</head>
<body>
<f:view>
  <h:form id="addForm">
     First Number:
     <h:inputText id="firstNumber"
        value="#{BackCalc.firstNumber}" required="true" />
     <h:message for="firstNumber" /> <br>
     Second Number:
     <h:inputText id="secondNumber"
        value="#{BackCalc.secondNumber}" required="true" />
     <h:message for="secondNumber" /> <br>
     Result:
     <h:outputText id="output" value="#{BackCalc.result}" /> <br>
     <h:commandButton id="submitAdd"
        action="#{BackCalc.add}" value="Add" />
  </h:form>
</f:view>
</body>
</html>
```

## backing bean para el pantallazo:

```
public class BackCalc {

  // atributos ligados a los UIComponents:
  private int firstNumber = 0;
  private int secondNumber = 0;
  private int result = 0;

  // model: podria ser la referencia a un EJB:
  private Calculator calculator = new Calculator();

  //. . . metodos set y get de atributos . . .

  // reaccion a evento sobre el boton:

  public String add() {
    result = calculator.add(firstNumber, secondNumber);
    return "success";
  }
}
```

- Validación automática:





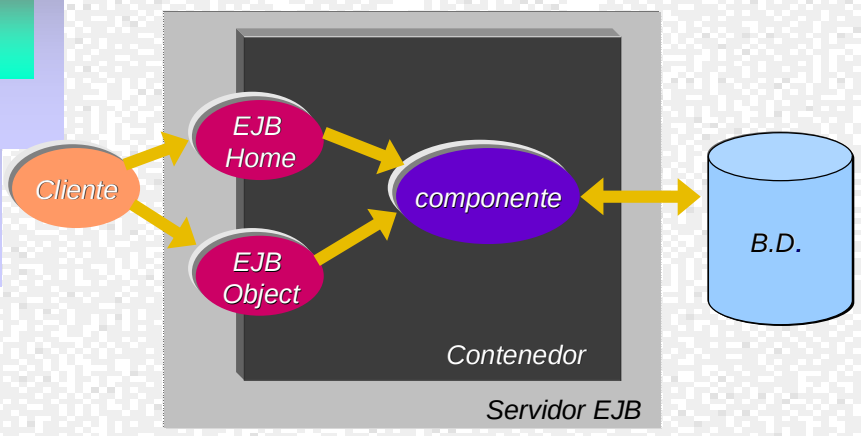

# Otros frameworks para components Web

- SpringMVC
- Webwork
- Shale
- Tapestry
- Wicket

# 5.

# Evolución de los frameworks para componentes de negocio

## Arquitectura básica de los EJB



## facilidades ofrecidas por los servidores EJB:

- manejo de nombres y localización de componentes
- generación de las clases “proxy” (i.e. stubs, skeletons)
- creación, interrupción y destrucción de componentes
- ejecución de componentes en threads
- persistencia
- soporte de transacciones distribuidas
- manejo eficiente de conexiones a BD (“pooling”)
- manejo de seguridad por roles y autenticación de usuarios



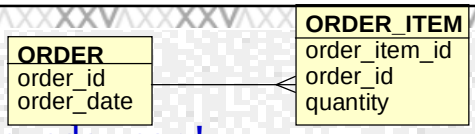
## tipos de componentes EJB:

- Session Bean (de sesión)
  - Cada instancia modela la sesión de un cliente ==> instancia NO COMPARTIDA
  - Estado no persistente (Transient)
- Entity Bean (de entidad)
  - Cada instancia modela una entidad de negocio
  - Estado persistente (reflejado en una BD)
  - Múltiples sesiones-clientes utilizan una instancia COMPARTIDA (identificada por llave primaria)
  - Sus métodos se realizan bajo transacciones controladas por el Contenedor
- Message Driven Bean (de mensajería)
  - Procesa mensajes asincrónicos (JMS)

# Framework estándar EJB 1.0:
*esfuerzo para el desarrollador*

- Interfaz remota de creación
- Interfaz remota de negocio
- Clase implementadora:
  - extiende clase base de componente: SessionBean o EntityBean
  - atributos de estado
  - métodos get() y set() para atributos de estado
  - métodos de negocio
  - métodos de intercepción: ejbCreate(), ejbFindByPrimaryKey(), ejbLoad(), ejbStore(), ...
- Descriptor J2EE:
  - valor de propiedades: participación en transacciones, tipo de persistencia, seguridad

37




- Ejb de entidad: no se puede usar !
  - modelo de persistencia automática demasiado restringido
- Ejb de sesión: JDBC demasiado dispendioso !

```
public Order loadOrder(Long orderId) throws DAOException {
    Order order = null;
    PreparedStatement ps = null; ResultSet rs = null;
    String queryOrder =
     " SELECT order_date FROM ORDER WHERE order_id = ? ";
    String queryOrderItems = " SELECT order_item_id, quantity "
     " FROM ORDER_ITEM WHERE order_id = ? ";
    try {
      dbConnection = datasource.getConnection();
      // consulta a tabla ORDER:
      ps = dbConnection.prepareStatement(queryOrder);
      ps.setLong(1, orderId.longValue());
      ps.executeQuery();
      rs = ps.getResultSet();
      rs.next();
      order = new Order();
      order.setOrderId(orderId);
      order.setOrderDate(rs.getDate(1));
      rs.close(); ps.close();
```

38

- continuación del ejemplo:

```
      // consulta a tabla ORDER_ITEM:
       ps = dbConnection.prepareStatement(queryOrderItems);
       ps.setLong(1, orderId.longValue());
       ps.executeQuery();
       rs = ps.getResultSet();
       List theItems = new ArrayList();
       while (rs.next()) {
         OrderItem orderItem = new OrderItem();
         orderItem.setOrderId(orderId);
         orderItem.setOrderItemId(rs.getString(1));
         orderItem.setQuantity (rs.getLong(2));
         theItems.add(orderItem);
       }
       order.setOrderItems(theItems);
     }
     catch (Exception e) {
       throw new Exception
          ("Problem loading an order: " + e.getMessage());
     }
     finally {
       rs.close();
       ps.close();
       dbConnection.close();
     }
     return order;
   }
```

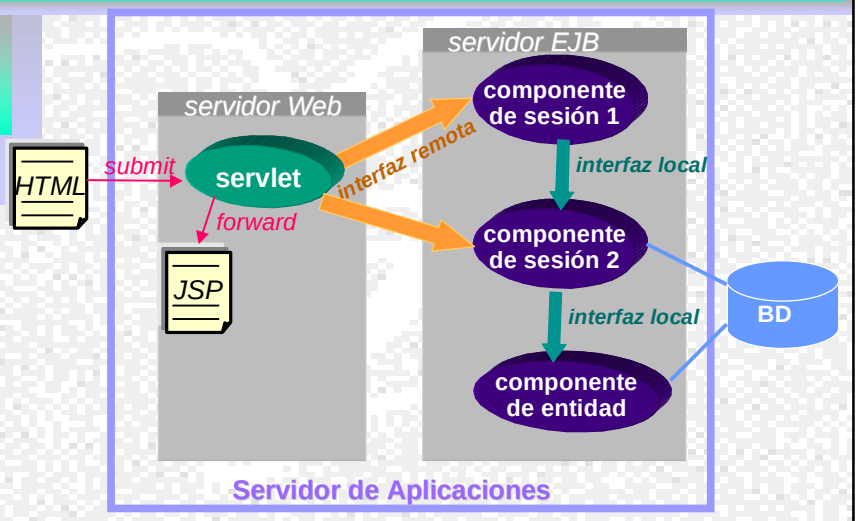


# Framework estándar EJB 2.1:

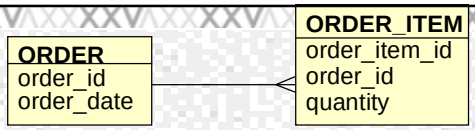
*demasiado esfuerzo para el desarrollador !*

- Interfaces de creación: local y/o remota
- Interfaces de negocio: local y/o remota
- Clase implementadora:
  - extiende clase base de componente: SessionBean o EntityBean
  - atributos de estado
  - métodos get() y set() para atributos de estado
  - métodos abstractos get() y set() para atributos de relación con otros componentes
  - métodos de negocio en términos de los métodos get() y set()
  - métodos de intercepción: ejbCreate(), ejbFindByPrimaryKey(), ejbLoad(), ejbStore(), ...
- Descriptor J2EE:
  - atributos de estado
  - relaciones con otros componentes
  - valor de propiedades: participación en transacciones, tipo de persistencia, seguridad

- Interfaces locales versus remotas
  - parámetros por referencia o por valor






- Ejb de entidad:
  - modelo de persistencia automática poderoso (maestro-detalle)
  - programación antinatural:

```
public abstract class OrderBean implements EntityBean {
  private Long orderId;
  private Date orderDate;

  // metodos abstractos get y set para atributos relacion CMR:
  public abstract List getOrderItems();
  public abstract void setOrderItems (List orderItems);
  ...
  // metodos de negocio: usan get y set CMR:
  public void addOrderItem(OrderItemLocal orderItem) {
    List orderItems = getOrderItems();
    orderItems.add(orderItem); // INSERT en ORDER_ITEM !!
  }
}
```

## Descriptor XML demasiado complejo !

- Interfaces, clases y atributos de estado (CMP) :

```
<ejb-jar>
   <enterprise-beans>
     <entity>
        <description>OrderBean</description>
        <ejb-name>OrderBean</ejb-name>
        <local-home>acme.OrderLocalHome</home>
        <local>acme.OrderLocal </remote>
        <ejb-class>acme.OrderBean</ejb-class>
        <persistence-type>Container</persistence-type>
        <prim-key-class>java.lang.Long</prim-key-class>
        <reentrant>False</reentrant>
        <cmp-version>2.x</cmp-version>
        <abstract-schema-name>Order</abstract-schema-name>
        <cmp-field>
           <field-name>orderDate</field-name>
         </cmp-field>
         <primkey-field>orderId</primkey-field>
     </entity>
     ....
   </enterprise-beans>
```

43

- Relaciones con otros EJBs (OrderBean con OrderItemBean):

```
<relationships>
  <ejb-relation>
    <description>Order items of order</description>
    <ejb-relationship-role>
       <ejb-relationship-role-name>OrderBean
       </ejb-relationship-role-name>
       <multiplicity>One</multiplicity>
       <relationship-role-source>
          <ejb-name>OrderBean</ejb-name>
       </relationship-role-source>
       <cmr-field>
         <cmr-field-name>orderItems</cmr-field-name>
         <cmr-field-type>java.util.List</cmr-field-type>
       </cmr-field>
    </ejb-relationship-role>

    <ejb-relationship-role>
       <ejb-relationship-role-name>OrderItemBean
       </ejb-relationship-role-name>
       <multiplicity>Many</multiplicity>
       <cascade-delete />
       <relationship-role-source>
          <ejb-name>OrderItemBean</ejb-name>
       </relationship-role-source>
    </ejb-relationship-role>
  </ejb-relation>
. . . . . .
```

44

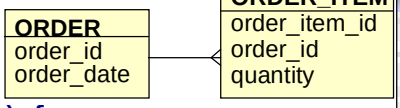
# Framework Spring combinado con Hibernate

- Movimiento de rebelión contra la complejidad de los EJB 2.x:
  - el retorno a los "**POJO**s" (Plain Old Java Objects) !
- Orientado a plugins que pueden conectarse a su contenedor:
  - soluciones de persistencia: Hibernate, JDO, TopLink, iBATIS, JDBC
  - soluciones de manejo de transacciones: JDBC, JTA, ...
  - soluciones de componentes web: SpringMVC, Struts, JSF, WebWork,...
- Descriptores y archivos de configuración de:
  - componentes y propiedades de cada nivel
  - asociación de componentes a tablas relacionales y descripción de relaciones maestro-detalle entre componentes
  - descriptores no estándares y complejos !



- Ej de componente en Spring (persistencia automática)

```
public class OrderItem implements serializable {
  private String orderItemId;
  private Long orderId;
  private long quantity;

  public String getOrderItemId() {
    return orderItemId;
  }

  public void setOrderItemId (String orderItemId) {
    this.orderItemId = orderItemId;
  }
  . . . // get y set para cada atributo
}
```




- Interfaz de negocio:

```
public interface MyApplicationFacade {
  OrderItem getOrderItem (String orderItemId);
  List getOrderItemsByOrder (Long orderId);
}
```

## Implementación de servicios a través de un objeto *dao* Hibernate:

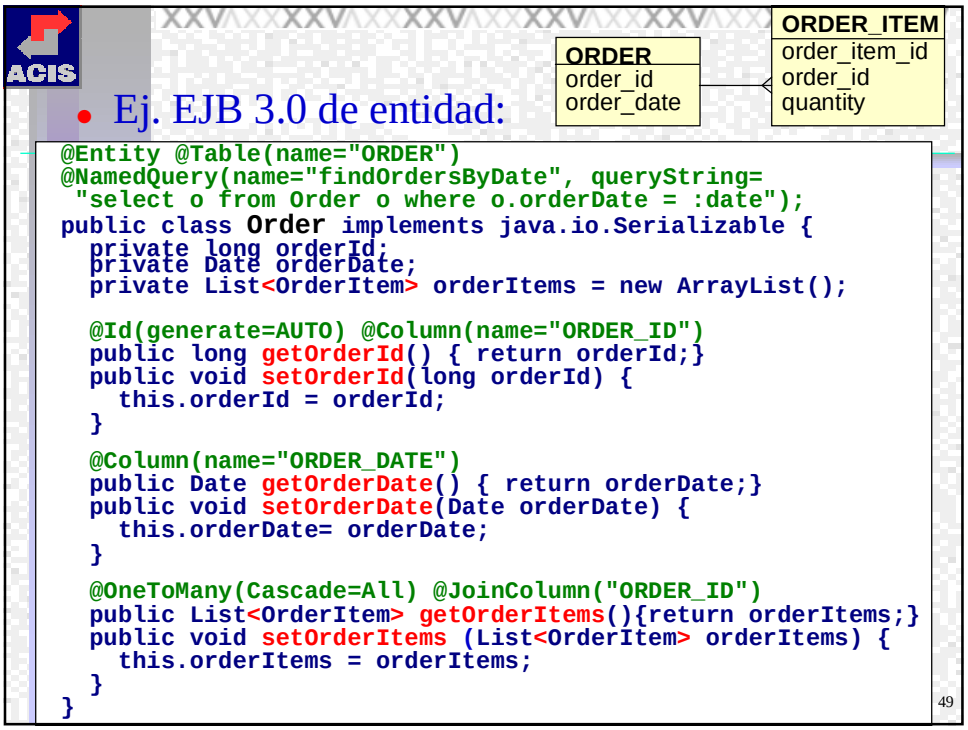
```
public class HibernateOrderItemDao extends HibernateDaoSupport
   implements OrderItemDao {
  private SessionFactory sessionFactory;

  public HibernateOrderItemDao () { }

  public setSessionFactory (SessionFactory sessionFactory) {
    this.sessionFactory = sessionFactory;
  }

  public OrderItem getOrderItem (String orderItemId)
                            throws DataAccessException{
    return (OrderItem)getHibernateTemplate()
      .load(OrderItem.class, orderItemId);   // SELECT en ORDER_ITEM
  }

  public List getOrderItemsByOrder (Long orderId) {
    return getHibernateTemplate()
        .find("from ORDER_ITEM where order_id = ? " ,
              orderId.longValue(), Hibernate.LONG);
  }
}
```



## Framework estándar EJB 3.0:

*reacción a la rebelión: reducir el esfuerzo del desarrollador*

- Interfaces de creación: NO
- Interfaces de negocio: NO para los de entidad
- Clase implementadora:
  - NO extiende clase base de componente: SessionBean o EntityBean
  - atributos de estado
  - métodos get() y set() para atributos de estado
  - atributos de relación con otros componentes
  - métodos get() y set() para atributos de relación con otros componentes
  - métodos de negocio
  - NO métodos de intercepción: ejbCreate(), ejbFindByPrimaryKey(), ejbLoad(), ejbStore(), ...
- Descriptor J2EE: NO es obligatorio
- En resumen:
  - se vuelve a un javaBean sencillo (POJO)
  - ***@anotaciones*** (J2SE 5.0) para cambiar valores por defecto de propiedades
  - más trabajo para el Contenedor, menos para el desarrollador
  - recoge experiencias de JDO, HIBERNATE, TOPLINK




- Ej. EJB 3.0 de entidad:

```
@Entity @Table(name="ORDER")
@NamedQuery(name="findOrdersByDate", queryString=
 "select o from Order o where o.orderDate = :date");
public class Order implements java.io.Serializable {
  private long orderId;
  private Date orderDate;
  private List<OrderItem> orderItems = new ArrayList();

  @Id(generate=AUTO) @Column(name="ORDER_ID")
  public long getOrderId() { return orderId;}
  public void setOrderId(long orderId) {
    this.orderId = orderId;
  }

  @Column(name="ORDER_DATE")
  public Date getOrderDate() { return orderDate;}
  public void setOrderDate(Date orderDate) {
    this.orderDate= orderDate;
  }

  @OneToMany(Cascade=All) @JoinColumn("ORDER_ID")
  public List<OrderItem> getOrderItems(){return orderItems;}
  public void setOrderItems (List<OrderItem> orderItems) {
    this.orderItems = orderItems;
  }
}
```



- Ej. EJB 3.0 de sesión que invoca al de entidad:
  - ilustra EntityManager: maneja ciclo de vida de EJBs de entidad
  - ofrece servicios a los servlets y jsp (parte web)

```
@Stateful
public class ShoppingCartBean implements ShoppingCart {
  @Inject EntityManager entityManager;

  private List itemsInCart;
  . . .

  public Order checkout() {
    Order order = new Order();
    order.setOrderDate(new Date());
    order.setOrderItems(this.itemsInCart);
    entityManager.persist(order); // INSERTs en ORDER y ORDER_ITEM
    return order;
  }

  public List<Order> findOrdersByDate (Date date) {
    return entityManager.createNamedQuery("findOrdersByDate")
      .setParameter("date", date).getResultList(); // SELECTs
  }

  public void updateOrder(Order order) {
    entityManager.merge (order); // UPDATEs en ORDER y ORDER_ITEM
  }
}
```

# 6. Conclusiones

- La arquitectura y frameworks J2EE es un tema en permanente evolución (con movimientos divergentes y convergentes)
- Dificultad de seguir la pista
- Tendencia hacia el modelaje pleno de objetos escondiendo la lógica SQL; simplificación de patrones
- Imposibilidad de desarrollar aplicaciones J2EE sin apoyarse en frameworks
- Decisión estratégica: adoptar los frameworks adecuados y las herramientas que los soporten.





# 7. Bibliografía


- Velocity: http://jakarta.apache.org/velocity/
- Framework STRUTS: http://jakarta.apache.org/struts/index.html
- JSF en Sun: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
- "JSF for nonbelievers" - Rick Hightower (ArcMind), Feb. 2005, http://www-128.ibm.com/developerworks/library/j-jsf1/
- Spring, Hibernate: "Better, Faster, Lighter Java" - Bruce A. Tate, O'Reilly 2004
- EJB 3.0: especificación en Sun: http://java.sun.com/products/ejb/docs.html
- "EJB 3.0 Preview" - Bill Burke (JBoss Chief Architect), Nov. 2004, http://java.sys-con.com/read/46975.htm
- presentaciones y artículos de congresos recientes:
  - JavaLobby, Dic. 2004: http://www.javalobby.org/av/javapolis/index.jsp
  - TheServerSide Java Symposium, Mayo 2005 http://www.theserverside.com/symposium/presentations.html#
  - JavaOne, Junio 2005: http://java.sun.com/javaone/sf/