Programación Avanzada

Patrón de Software MVC

Esp. Ing. César Aranda

unidatos@gmail.com

Licenciatura en Informática y Desarrollo de Software

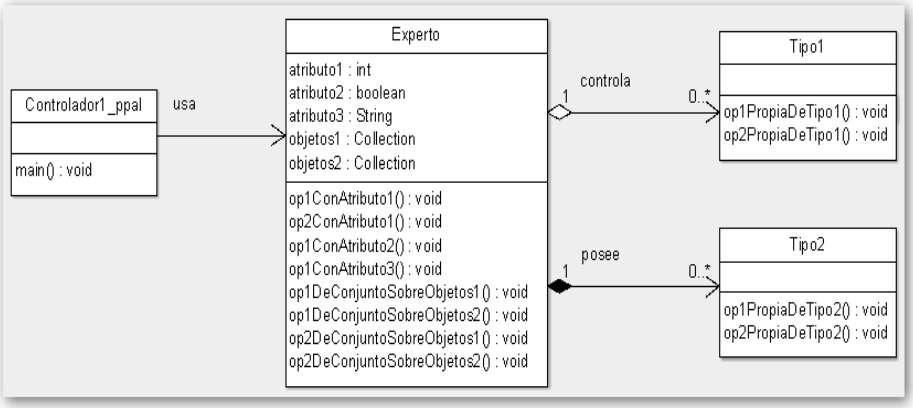
v.2014

Patrón Experto (GRASP)

- El objetivo es asignar una responsabilidad al Experto en Información, es decir a la clase que cuenta con la información necesaria para cumplir la responsabilidad.
- Resulta aplicable cuando:
  - Se desea sencillez en el diseño, ya que responde a un principio básico del paradigma de objetos.
  - Se desea centralizar en una clase, el manejo de información distribuida en varias clases de objetos.
  - Se desea conservar el encapsulamiento y mantener un bajo acoplamiento, para favorecer la obtención de un sistema más robusto y de fácil mantenimiento.
  - Hay que aumentar tanto la cohesión como la facilidad de comprensión y mantenimiento posterior

2

Estructura general



3

Patrón Controlador (GRASP)

- Tiene la finalidad de manejar los eventos externos del sistema
- Resulta aplicable cuando se desea:
  - Desacoplar la interfaz de usuario del tratamiento de eventos de alto nivel del sistema generados por un actor externo.
  - Centralizar un conjunto de operaciones asociadas a eventos del sistema.
  - Obtener una clase reutilizable para colaborar en diferentes vistas.
  - Garantizar que la empresa o los procesos de dominio sean manejados por la capa de objetos del dominio y no por la de interfaz de usuario.
  - Asegurar que las operaciones del sistema sigan una secuencia legal o controlar el estado actual de la actividad y las operaciones en el caso de uso subyacente.

4

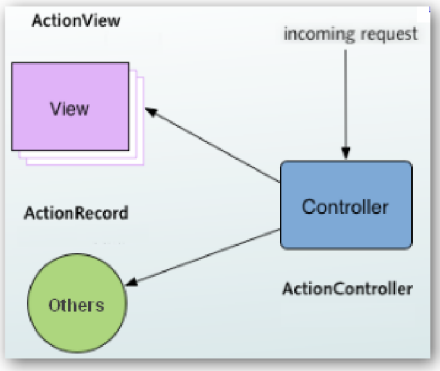
¿Qué es un Controlador?

- Es un objeto de interfaz no destinado al usuario que se encarga de manejar un evento del sistema.
- Define la forma de operación de la interfaz de usuario.
- Una clase que:
  - representa al sistema completo, a la empresa u organización global. Puede considerarse como un controlador de Fachada
  - representa una parte activa del mundo real que participa en la tarea y desencadena cierto evento. Por lo que puede considerarse como un controlador de Rol o de Tareas
  - representa un manejador auxiliar o artificial para cierto conjunto de eventos del sistema asociados al hacer del usuario, generalmente denominado como un caso de uso gestionar, administrar, manejar, etc. Por lo que puede considerarse como un controlador del Caso de Uso

5

Estructura general

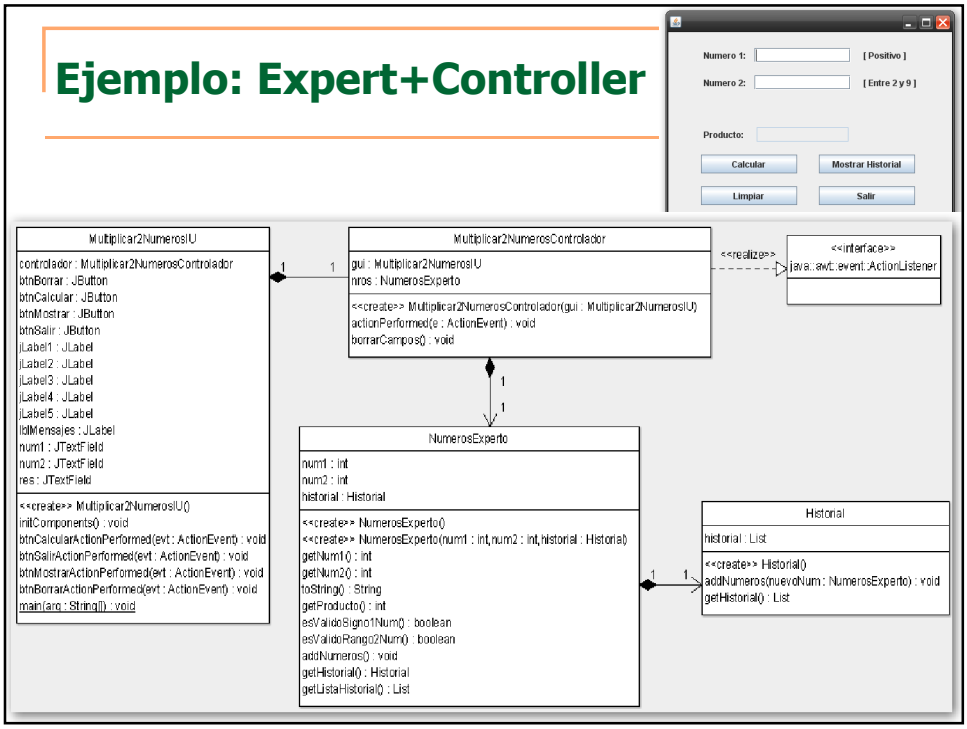
- **View**: clase que representa a la interfaz del usuario, donde se producen los eventos del sistema.
- **Controller:** clase que modela el comportamiento con la respuesta a los eventos y se comunica con el resto (o un subconjunto) de los objetos de las capas internas del modelo.
- **Others:** clases con otros controladores o las definiciones propias del sistema.



Observación: este patrón es básico y es usado por muchos otros modelos, como son los caso del patrón J2EE Front Controller en J2EE o de la arquitectura MVC.

6

Ejemplo: Expert+Controller



Ejemplo: detalle del Experto

```
5   public class NumerosExperto {
6       private int num1;
7       private int num2;
8       private Historial historial;
9
10      public NumerosExperto() {
11          this.historial = new Historial();
12      }
13
14      public NumerosExperto(int num1, int num2, Historial historial) {
15          this.num1 = num1;
16          this.num2 = num2;
17          this.historial = historial;
18      }
```

```
6   public class Historial {
7       private List<NumerosExperto> historial;
8
9       public Historial() {
10          historial = new ArrayList<NumerosExperto>();
11      }
12
13      public void addNumeros(NumerosExperto nuevoNum) {
14          historial.add(nuevoNum);
15      }
16
17      public List<NumerosExperto> getHistorial() {
18          return historial;
19      }
20  }
```

```
28      @Override
29      public String toString() {
30          return "num 1 \t= \t" + this.num1 + "\nnum 2 \t= \t" + this.n
31      }
32
33      public int getProducto() {
34          return this.num1 * this.num2;
35      }
36
37      public boolean esValidoSigno1Num() {...}
44
45      public boolean esValidoRango2Num() {...}
52
53      public void addNumeros() {
54          historial.addNumeros(this);
55      }
56
57      public Historial getHistorial() {
58          return historial;
59      }
60      public List<NumerosExperto> getListaHistorial() {
61          return historial.getHistorial();
62      }
63  }
```

```
3   public class Multiplicar2NumerosIU extends javax.swing.JFrame {
4
5       Multiplicar2NumerosControlador controlador;
6
7       public Multiplicar2NumerosIU() {
8           initComponents();
9           controlador = new Multiplicar2NumerosControlador(this);
10      }
11
12      @SuppressWarnings("unchecked")
13      Generated Code
148
149     private void btnCalcularActionPerformed(java.awt.event.ActionEv
150         controlador.actionPerformed(evt);
151     }
152
153     private void btnSalirActionPerformed(java.awt.event.ActionEvent
154         controlador.actionPerformed(evt);
```

Ejemplo: detalle del Controlador

```
6  public class Multiplicar2NumerosControlador implements ActionListener {
7
8      private Multiplicar2NumerosIU gui;
9      private NumerosExperto nros;
10
11     public Multiplicar2NumerosControlador(Multiplicar2NumerosIU gui) {
12         this.gui = gui;
13         // Creación de un objeto experto vacío para contar con el objeto Historial
14         nros = new NumerosExperto();
15     }
16
17     public void actionPerformed(ActionEvent e) {
18         String comando = e.getActionCommand();
19         if (comando.equals("Calcular")) {
20             // Se asume que se ha ingresado un número con caracteres validos
21             int num1 = Integer.parseInt(gui.num1.getText());
22             int num2 = Integer.parseInt(gui.num2.getText());
23
24             nros = new NumerosExperto(num1, num2, nros.getHistorial());
25
26             if (nros.esValidoSigno1Num()) {
27                 if (nros.esValidoRango2Num()) {
28                     int result = nros.getProducto();
29                     nros.addNumeros();
30                     gui.res.setText(String.valueOf(result));
31                 } else {
32                     gui.lblMensajes.setText("2° numero fuera de rango");
```

El controlador actúa como oyente de eventos (implementa la interface ActionListener)

9

Patrón MVC

- También llamado Arquitectura Modelo-Vista-Controlador
- ¿Es un patrón de diseño o de arquitectura?
- Los patrones arquitectónicos se enfocan en aspectos más amplios que los patrones de diseño.
- Como patrón arquitectónico expresa un esquema de organización estructural esencial para un sistema de software, que consta de subsistemas, sus responsabilidades e interrelaciones. Encarna diferentes atributos de calidad.
- Como patrón de diseño expresa una forma particular de disponer las clases y de manejar sus interacciones.

Ing. César Omar Aranda -Lic. Enrique Ruiz 10

Clasificación de MVC

- El estilo fue descrito por primera vez en 1979 por Trygve Reenskaug, entonces trabajando en Smalltalk-80 en laboratorios de investigación de Xerox.
- Según Buschmann, pertenece a la categoría de Sistemas Interactivos (Patrones de Arquitectura)

"Pattern Oriented Software Architecture", 1996

- Según Fowler: pertenece a categoría de Patrones de Presentación Web (Patrones de Arquitectura)

"Pattern of Enterprise Application Architecture", 2003

- Según Stelting & Maassen: pertenece a la categoría de Patrones de Sistema (Patrones de Arquitectura/Diseño)

"Applied Java Patterns", 2001

Ing. César Omar Aranda -Lic. Enrique Ruiz 11

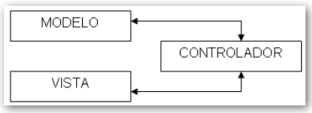
Elementos de MVC

- Modelo (Model): Encapsula los datos y la funcionalidad esencial. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.
- Vista (View): Despliega la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.
- Controlador (Controller): Recibe y maneja las entradas, usualmente como eventos que codifican los movimientos o pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio ("service requests") para el modelo o la vista.
- Este patrón divide una aplicación interactiva en 3 áreas: procesamiento, salida y entrada.
  - Las vistas y los controladores juntos componen la interfaz con el usuario.
  - Desacopla el modelo de las vistas.
  - El mecanismo de cambio-propagación asegura la consistencia de la interfaz con el modelo.

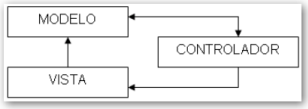
Ing. César Omar Aranda -Lic. Enrique Ruiz 12

Variantes de MVC

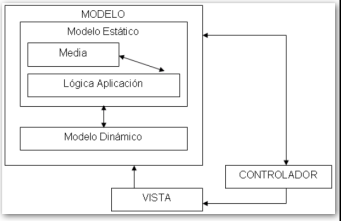
- Variante I
  - No existe ninguna comunicación entre el Modelo y la Vista y esta última recibe los datos a mostrar a través del Controlador.



- Variante II
  - Se desarrolla una comunicación entre el Modelo y la Vista, donde esta última al mostrar los datos los busca directamente en el Modelo, dada una indicación del Controlador, disminuyendo el conjunto de responsabilidades de este último.



- Variante III
  - Se diversifican las funcionalidades del Modelo teniendo en cuenta las características de las aplicaciones multimedia, donde tienen un gran peso los tipos de medios utilizados en éstas.



Ing. César Omar Aranda -Lic. Enrique Ruiz 13

Aplicabilidad MVC

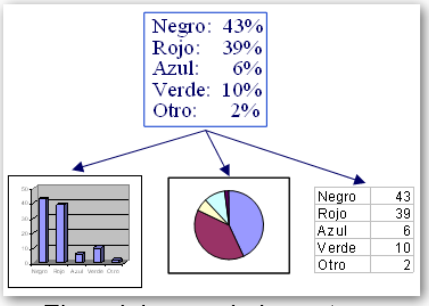
Puede aplicarse este patrón en aquellos casos en que:

- Se requieren aplicaciones interactivas con interfaces humano-computador cambiantes y flexibles.
- Las interfaces de usuario son modificadas con mucha frecuencia.
- Hay cambios en la funcionalidad que deben reflejarse en las interfaces.
- Existen interfaces a medida para ciertos usuarios.
- Se presentan diferentes paradigmas de interfaz
  - digitar información mediante consola de comandos
  - seleccionar íconos en interfaces gráficas

Ing. César Omar Aranda -Lic. Enrique Ruiz 14

Ejemplo: Sistema de Encuestas

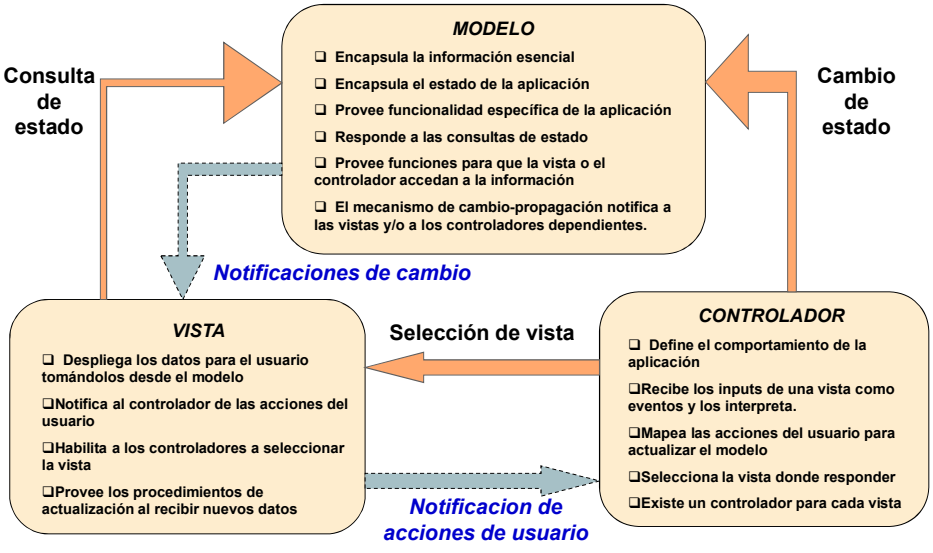
- Los usuarios votan a través de una interfaz gráfica.
- La información se muestra con distintos formatos.
- Los cambios por votación deben reflejarse en forma inmediata.
- Debe poderse integrar otras formas de presentación de los resultados tales como la distribución de votaciones por horario o por región.
- Las presentaciones deben ser portables.



- El modelo guarda los votos acumulados para cada opción y permite que las vistas accedan a los números de votos.
- Hay vistas de diferente tipo: gráfico de barras, de torta o tabla.

Ing. César Omar Aranda -Lic. Enrique Ruiz 15

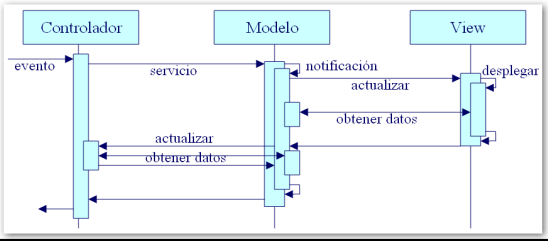
Estructura General de MVC



Ing. César Omar Aranda -Lic. Enrique Ruiz 16

Comportamiento de MVC

1. El usuario realiza una acción en la interfaz y dispara un evento.
2. El controlador trata el evento de entrada.
   - Previamente se ha registrado
3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta).
4. Se genera una nueva vista. La vista toma los datos del modelo.
   - El modelo no tiene conocimiento directo de la vista
5. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo.



Ing. César Omar Aranda -Lic. Enr 17

Implementaciones

Utilizado en múltiples frameworks

- Java Swing
- Java Enterprise Edition (J2EE)
- XForms (Formato XML estándar del W3C para la especificación de un modelo de proceso de datos XML e interfaces de usuario como formularios web)
- GTK+ (escrito en C, toolkit creado por Gnome para construir aplicaciones gráficas, inicialmente para el sistema X Window)
- ASP.NET MVC Framework (Microsoft)
- Google Web Toolkit (GWT, para crear aplicaciones Ajax con Java)
- Apache Struts (framework para aplicaciones web J2EE)
- Ruby on Rails (framework para aplicaciones web con Ruby)
- ...

Ing. César Omar Aranda -Lic. Enrique Ruiz 18

Relaciones de MVC

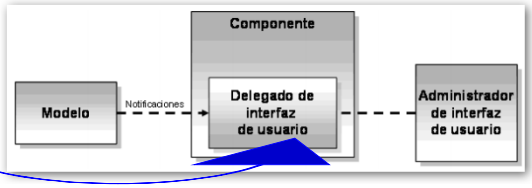
- Las implementaciones de MVC utilizan varios de los siguientes patrones:
  - Observer
  - Composite
  - Strategy
  - Decorator
  - Factory method
  - Command
  - Controller
- Todos ellos son patrones de diseño.

Ing. César Omar Aranda -Lic. Enrique Ruiz 19

Diseño cuasi-MVC de Java Swing

- Los primeros prototipos Swing siguieron un a separación MVC tradicional: cada componente tenía un objeto modelo y un objeto delegado con su look-and-feel para separar los objetos de vista y controlador.
- Rápidamente se descubrió que esa separación no trabajaba bien en términos prácticos porque las partes de la vista y del controlador de un componente requerían un estrecho acoplamiento y resultaba muy difícil escribir controladores genéricos que desconocieran los aspectos específicos de la vista.
- Por ello, se colapsaron las 2 entidades en cada componente de interfaz usuaria:

Objeto Delegado de Interfaz Usuaria
(UI Delegate)



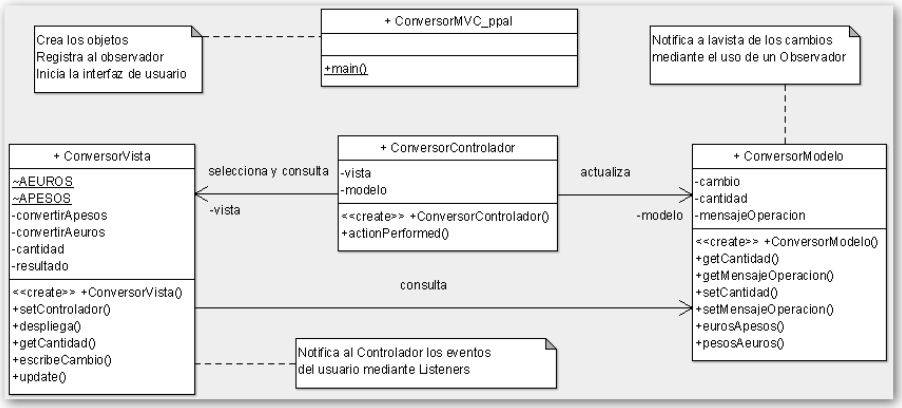
Ing. César Omar Aranda -Lic. Enrique Ruiz 20

Uso de Swing para Diseños MVC

- Modelo
  - El modelo lo realiza el desarrollador
- Vista
  - Conjunto de objetos de clases orientadas a Interfaces Usuarias que heredan de java.awt.Component
- Controlador
  - El controlador es el thread de tratamiento de eventos, que captura y propaga los eventos a la vista y al modelo
  - Clases de tratamiento de los eventos (a veces como clases anónimas) que implementan interfaces del tipo EventListener (ActionListener, MouseListener, WindowListener, etc.) o clases adaptadoras.
  - Eventualmente, objetos Observadores para realizar cambios en la/s vista/s.

Ing. César Omar Aranda -Lic. Enrique Ruiz 21

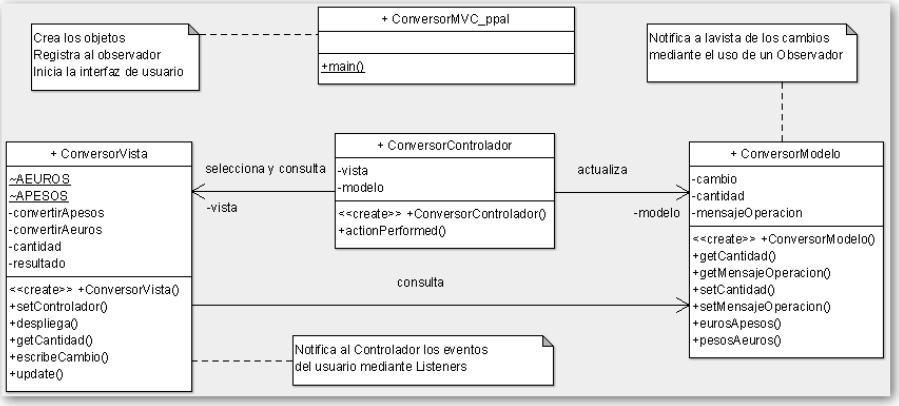
Ejemplo 1 (simple)



Ing. César Omar Aranda -Lic. Enrique Ruiz 22

Ejemplo 1 + Patrones

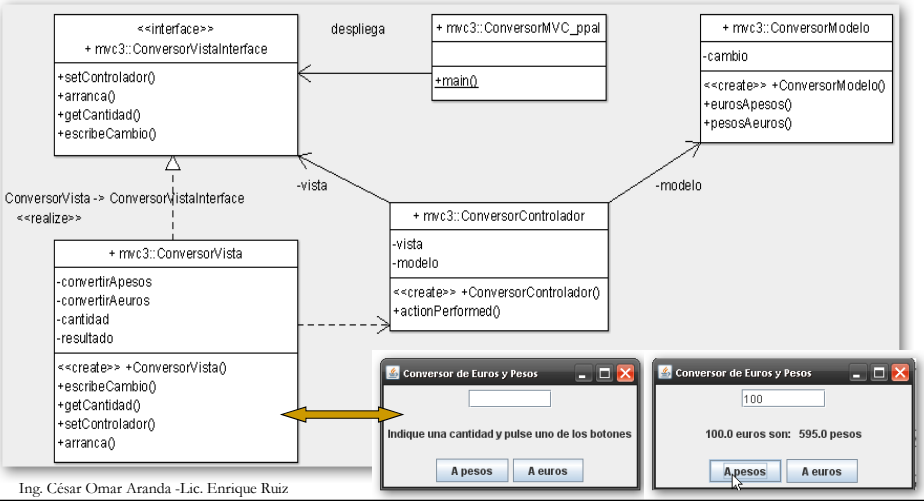
- MVC como Patrón de Diseño con Observador de cambios en el modelo



Ing. César Omar Aranda -Lic. Enrique Ruiz 23

Ejemplo 2

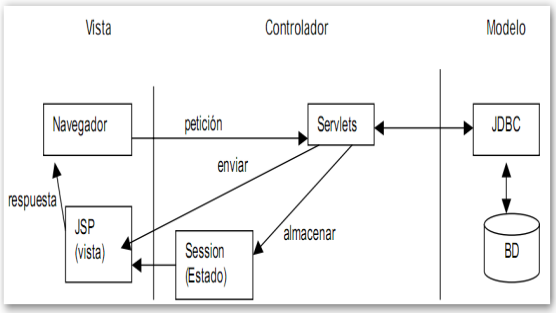
- MVC como Patrón de Diseño con Interface para Diferentes vistas y sin Observador de Modelo

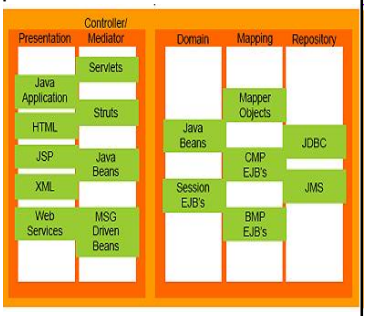


Ing. César Omar Aranda -Lic. Enrique Ruiz

Estructura MVC como Arquitectura

- Clientes basados en Web (browsers): JSP genera las vistas, Servlets propone controladores, y los componentes EJB el modelo.



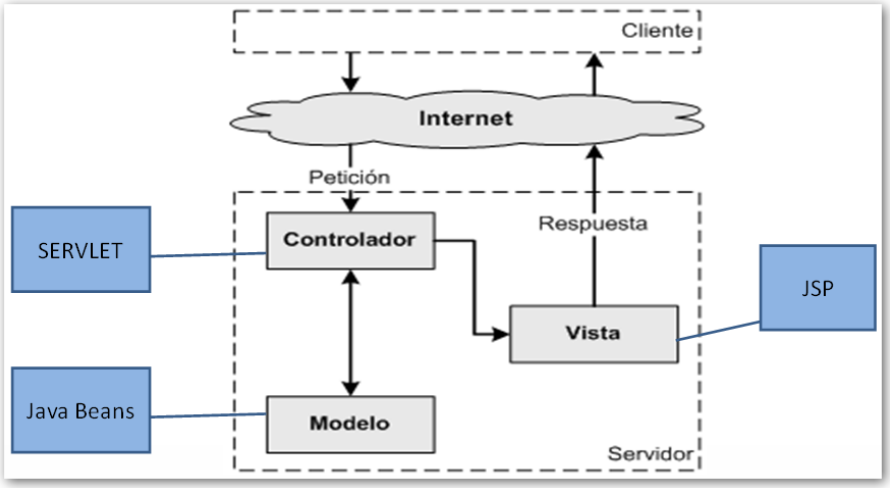


- Controlador centralizado: En lugar de tener múltiples servlets como controladores, se usa un Servlet principal para hacer el control más manejable: Patrón Front Controller.
- Clientes Inalámbricos: como en el caso de teléfonos celulares.

Ing. César Omar Aranda -Lic. Enrique Ruiz 25

Ejemplo 3

- MVC como Patrón de Arquitectura con diferentes tecnologías Java



Ing. César Omar Aranda -Lic. Enrique Ruiz 26

Referencias

- GAMMA, Erich, HELM, Richard y otros (2003): Patrones de Diseño. Pearson Education.
- STELTING, Stephen, MAASSEN, Olav (2003): Patrones de diseño aplicados a Java. Editorial Pearson.
- LARMAN, Craig (2003): UML y patrones. 2ª edición. Prentice Hall.
- http://www.mindview.net/Seminars/ThinkingInPatterns/
- http://es.wikipedia.org/wiki/GRASP
- http://jorgesaavedra.wordpress.com/category/patrones-grasp/
- http://www.practicasdesoftware.com.ar/2011/03/patrones-grasp/
- http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html
- http://java.sun.com/blueprints/patterns/MVC-detailed.html
- http://www.oracle.com/technetwork/java/mvc-140477.html
- http://caraballomaestre.blogspot.com/2009/02/arquitectura-j2ee-patron-mvc.html
- http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html
- http://es.debugmodeon.com/articulo/el-patron-mvc