2. Patrones de arquitectura multicapa

Índice

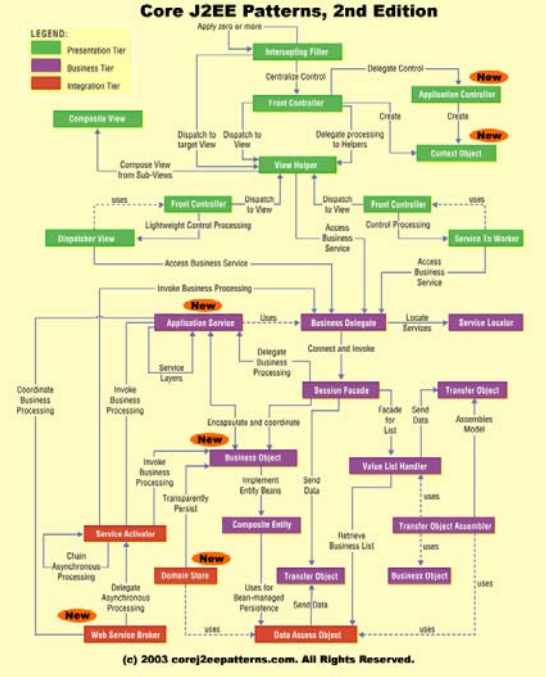
Referencias

- Alur, D., Malks, D., Crupi. J. *Core J2EE Design Patterns: Best Practices and Design Strategies. 2nd Edition.* Prentice Hall, 2003
- Fowler, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2003

Introducción

- En este tema veremos los patrones de la *arquitectura multicapa*
- Como ya sabemos, están agrupados en tres capas:
  - Presentación
  - Negocio
  - Integración

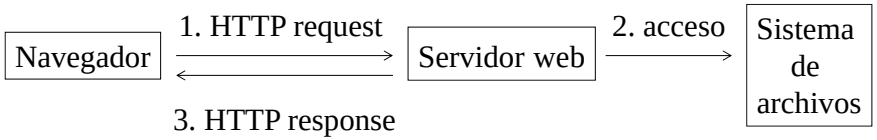
Catálogo de patrones multicapa



Introducción

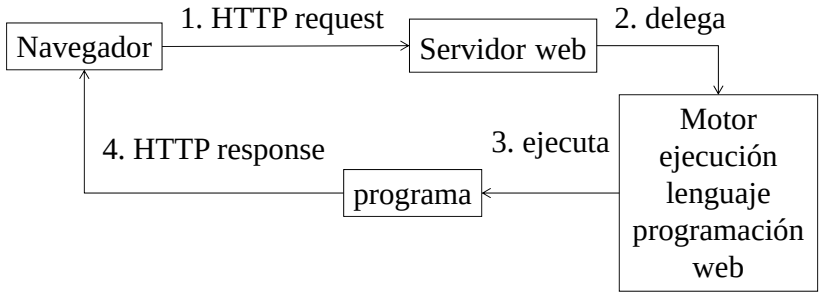
- Un servidor web utiliza el protocolo HTTP para recibir y enviar información a través de *requests/responses* (peticiones/respuestas)

Introducción



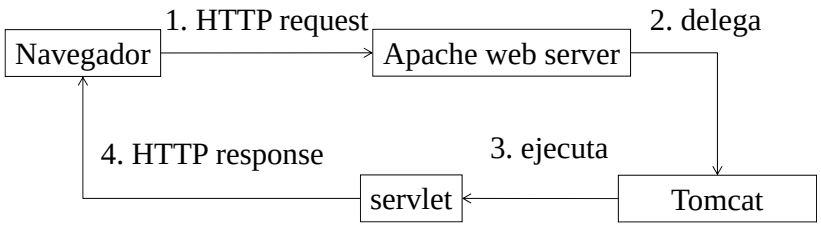
Esquema de funcionamiento de un servidor web

Introducción



Esquema de funcionamiento de un servidor web extendido para ejecutar código de propósito general

Introducción



Ejemplo concreto de servidor y motor de ejecución

Introducción

- Ejemplo*:


```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

*http://www.roseindia.net/servlets/HelloWorld.shtml

Introducción

```
public class HelloWorld extends HttpServlet{
  public void doGet(HttpServletRequest request,
  HttpServletResponse response)
  throws ServletException,IOException{
  response.setContentType("text/html");
  PrintWriter pw = response.getWriter();
  pw.println("<html>");
  pw.println("<head><title>Hello
World</title></title>");
  pw.println("<body>");
  pw.println("<h1>Hello World</h1>");
  pw.println("</body></html>");
  }
}
```


Introducción

```
<web-app>
 <servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>Hello</servlet-name>
 <url-pattern>/HelloWorld</url-pattern>
 </servlet-mapping>
</web-app>
```

Tipo de aplicación	Solución	Uso de marcos	Ejemplos implementación J2EE		
			Pres.	Negocio	Integración
Empresarial	Arquitectura multicapa	No	JSP	SAs POJOs + transfers	DAOs POJOs
		Sí	JSF	SAs POJOs + entidades	JPA
+ Lógica distribuida	+ RPC (Remote Procedure Call)	No	JSP	SAs POJOs + RMI + transfers	DAOs POJOs
		Sí	JSF	EJBs de sesión + entidades	JPA
+ Plataformas heterogéneas	+ SOA	No	JSP	SAs POJOs + transfers + JAX WS / JAX RS	DAOs POJOs
		Sí	JSF	EJBs de sesión JAX WS / JAX RS + entidades	JPA

Solución aplicable en base a los requisitos de aplicación

Capa presentación Intercepting filter

- Propósito
  - Se desea interceptar y manipular peticiones y respuestas antes y después del procesamiento de la petición
- También conocido como
  - Filtro de interceptación

Capa presentación Intercepting filter

- Motivación
  - A veces es necesario preprocesar la petición antes de enviarla a negocio. Por ejemplo:
    - Validar si el usuario tiene una sesión válida
    - Comprobar si el navegador es soportado
    - Comprimir un flujo de respuesta
  - Podría utilizarse lógica if/then/else, pero el problema es similar al de la cadena de responsabilidad

Capa presentación Intercepting filter

- Contexto
  - Se desea un procesamiento común y centralizado de los peticiones
  - Se desea tener componentes de pre y posprocesado con un bajo acoplamiento con negocio para facilitar este procesamiento
  - Se desean componentes de pre y posprocesado independientes y autocontenidos para facilitar su reutilización

Capa presentación
Intercepting filter

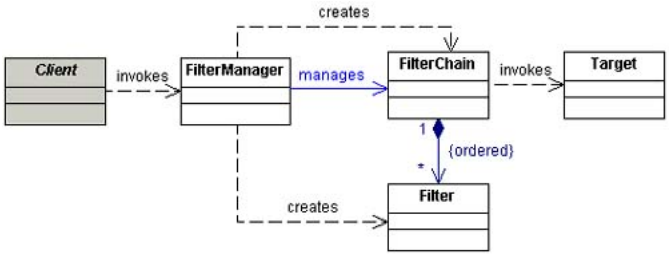
- Solución
  - Utilizar un intercepting filter como filtro conectable para pre y posprocesar peticiones y respuestas
  - Un gestor de filtros combina filtros débilmente acoplados en una cadena, delegando el control al filtro adecuado

Capa presentación
Intercepting filter

  - Así, se puede añadir, eliminar y combinar estos filtros de distintas formas sin variar código existente

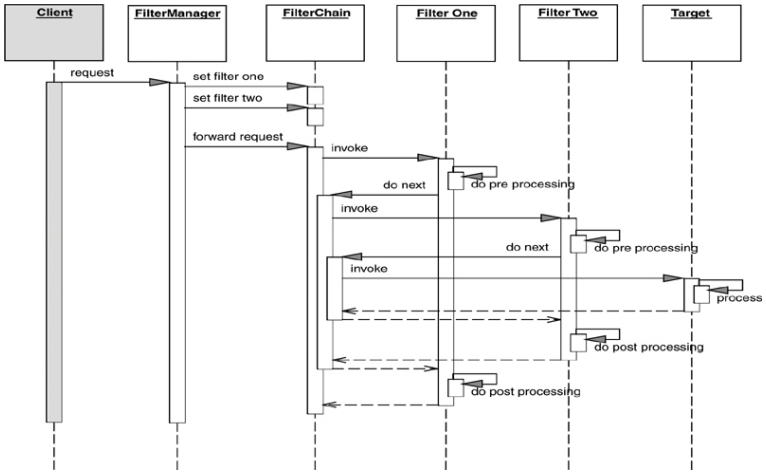
Capa presentación
Intercepting filter

- Descripción



Estructura del patrón intercepting filter

Capa presentación



Interacción en patrón intercepting filter

Capa presentación
Intercepting filter

- Consecuencias
  - Ventajas
    - Centraliza el control mediante manejadores con bajo acoplamiento
    - Mejora la reusabilidad
    - Permite una configuración flexible y declarativa
  - Inconvenientes
    - El compartir información entre filtros es ineficiente

Capa presentación
Intercepting filter

- Ejemplo

```
public final class HitCounterFilter implements
Filter {
   private FilterConfig filterConfig = null;
   public void init(FilterConfig filterConfig)
      throws ServletException {
      this.filterConfig = filterConfig;
   }
   public void destroy() {
      this.filterConfig = null;
   }
```


Capa presentación
Intercepting filter

```
public void doFilter(ServletRequest request,
      ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
      if (filterConfig == null)
         return;
      StringWriter sw = new StringWriter();
      PrintWriter writer = new PrintWriter(sw);
      Counter counter = (Counter)filterConfig.
         getServletContext().
         getAttribute("hitCounter");
      writer.println();
      writer.println("===============");
```


Capa presentación
Intercepting filter

```
      writer.println("The number of hits is: " +
         counter.incCounter());
      writer.println("===============");

      // Log the resulting string
      writer.flush();
      filterConfig.getServletContext().
         log(sw.getBuffer().toString());
      ...
      chain.doFilter(request, wrapper);
      ...
   }
}
```

Capa presentación
Controlador Frontal

- Propósito
  - Proporciona un punto de acceso para el manejo de las peticiones de la capa de presentación
- También conocido como
  - Front controller

Capa presentación
Controlador Frontal

- Motivación
  - Se desea evitar lógica de control duplicada
  - Se desea aplicar una lógica común a distintas peticiones
  - Se desea separar la lógica de procesamiento del sistema de la vista
  - Se desea tener puntos de acceso centralizado y controlado al sistema

Capa presentación
Controlador Frontal

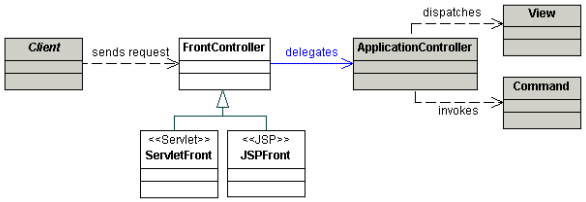
- Contexto
  - Se quiere tener un punto inicial de contacto para manejar las peticiones, centralizando la lógica de control y manejando las actividades de manejo de peticiones

Capa presentación
Controlador Frontal

- Solución
  - Utilizar un controlador frontal como el punto inicial de contacto para manejar todas las peticiones
  - El controlador frontal centraliza la lógica de control que podría estar duplicada de no existir dicho controlador y gestiona las actividades básicas de manejado de las peticiones

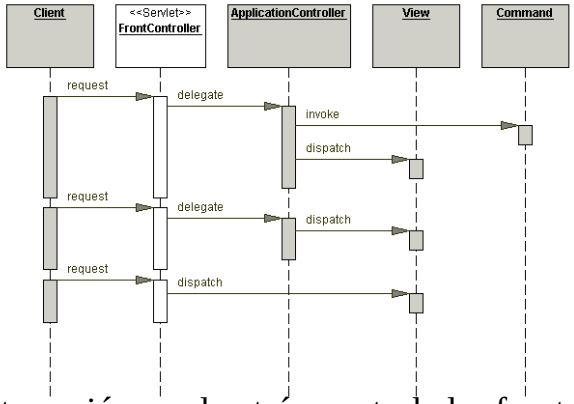
Capa presentación
Controlador Frontal

- Descripción



Estructura del patrón controlador frontal

Capa presentación
Controlador Frontal



Interacción en el patrón controlador frontal

Capa presentación
Controlador Frontal

- Consecuencias
  - Ventajas:
    - Centraliza el control
    - Mejora la gestión de la aplicación
    - Mejora la reutilización
    - Mejora la separación de roles

Capa presentación
Controlador Frontal

- Código de ejemplo

```
public class FrontController extends HttpServlet
{
 protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws
ServletException, java.io.IOException {

      processRequest(request, response);

    }
```

Capa presentación
Controlador Frontal

```
protected void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, java.io.IOException
{

      processRequest(request, response);
    }
```


Capa presentación
Controlador Frontal

```
protected void processRequest(HttpServletRequest
request, HttpServletResponse response) throws
ServletException, java.io.IOException {
      String page;
      ApplicationResources resource =
ApplicationResources.getInstance();
      try {
        RequestContext requestContext =
            new RequestContext(request, response);
```


Capa presentación
Controlador Frontal

```
ApplicationController applicationController = new
            ApplicationControllerImpl();
ResponseContext responseContext =
applicationController.handleRequest(requestContext);
applicationController.handleResponse(
             requestContext, responseContext);
      } catch (Exception e) {
LogManager.logMessage("FrontController:exception : " +
            e.getMessage());
request.setAttribute(resource.getMessageAttr(),
            "Exception occurred : " + e.getMessage());
        page = resource.getErrorPage(e);
```


Capa presentación
Controlador Frontal

```
dispatch(request, response, page);
      }
    }
//sólo se utiliza esta función si hay error
protected void dispatch(HttpServletRequest request,
HttpServletResponse response, String page)
throws javax.servlet.ServletException, java.io.IOException
{
 RequestDispatcher dispatcher = this.getServletContext().
          getRequestDispatcher(page);
  dispatcher.forward(request, response);
    }
}
```

Capa presentación
Controlador de aplicación

- Propósito
  - Se desea centralizar y modularizar la gestión de acciones y de vistas
- También conocido como
  - Application controller

Capa presentación
Controlador de aplicación

- Motivación
  - Se desea reutilizar el código de gestión de vistas y acciones
  - Se desea mejorar la extensibilidad de el manejo de peticiones (p.e. añadir casos de uso a una aplicación incrementalmente)

Capa presentación
Controlador de aplicación

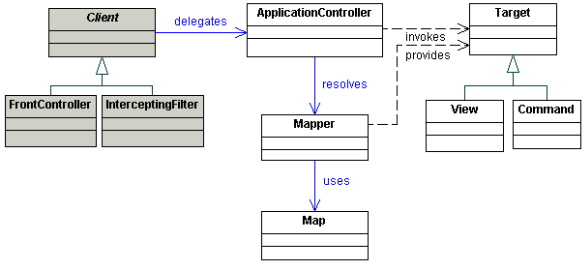
- Se desea mejorar la modularidad del código y la mantenibilidad, facilitando al extensión de la aplicación y la prueba del código de manejo de peticiones de manera independiente del contenedor web

Capa presentación
Controlador de aplicación

- Contexto
  - Se quiere centralizar la recuperación e invocación de componentes de procesamiento de las peticiones, tales como comandos y vistas
- Solución
  - Utilizar un controlador de aplicación para centralizar la recuperación e invocación de componentes de procesado de peticiones, tales como comandos y vistas

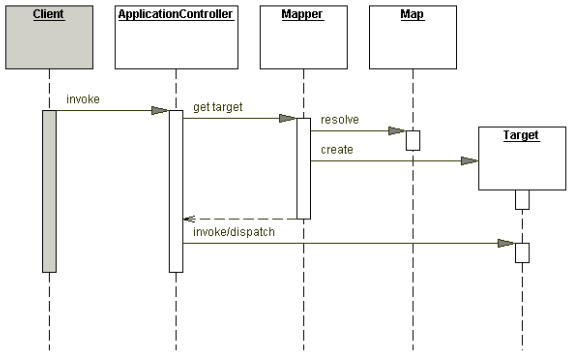
Capa presentación
Controlador de aplicación

- Descripción



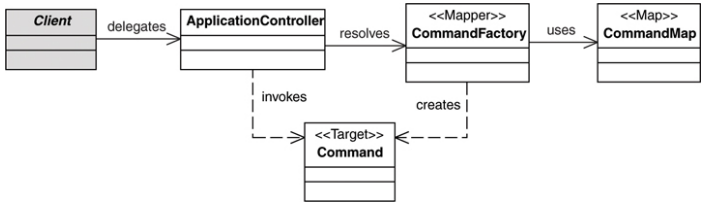
Estructura del patrón controlador de aplicación

Capa presentación
Controlador de aplicación



Interacción en el patrón controlador de aplicación

Capa presentación
Controlador de aplicación

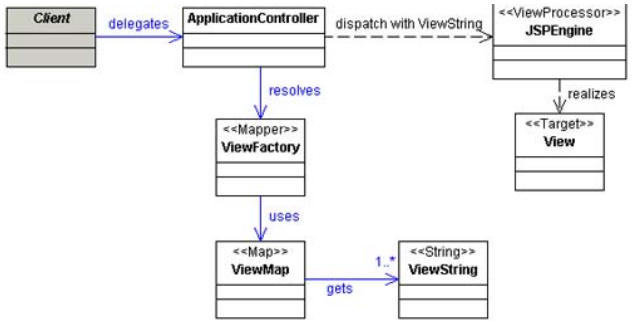


Controlador de aplicación y comando

Capa presentación
Controlador de aplicación

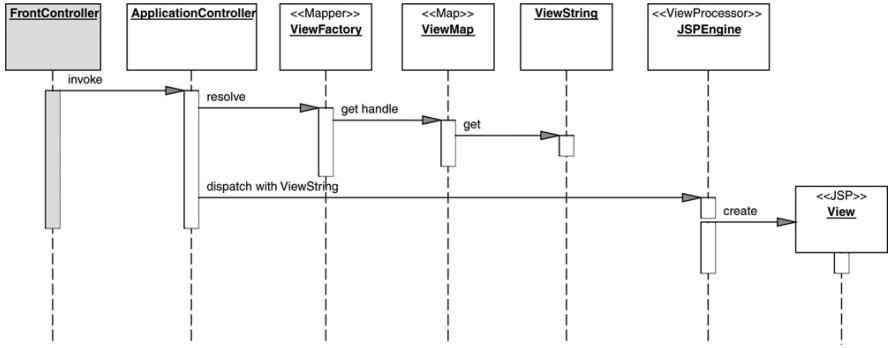
- La redirección a la vista puede
  - Delegarse al controlador frontal
  - Incluirse en el action/comando
  - Delegarse al controlador de aplicación

Capa presentación
Controlador de aplicación



Redirección a vistas por parte del controlador de aplicación

Capa presentación
Controlador de aplicación



Redirección a vistas por parte del controlador de aplicación

Capa presentación
Controlador de aplicación

- Consecuencias
  - Ventajas
    - Mejora la modularidad
    - Mejora la reutilización
    - Mejora la extensibilidad
  - Inconvenientes
    - Aumenta el número de objetos involucrados
    - En aplicaciones grandes puede llegar a crecer mucho

Capa presentación
Controlador de aplicación

- Código de ejemplo

```
interface ApplicationController {
ResponseContext handleRequest(RequestContext
requestContext);
void  handleResponse(RequestContext requestContext,
ResponseContext responseContext);
}
```

Capa presentación
Controlador de aplicación

```
class WebApplicationController implements
ApplicationController {

public ResponseContext handleRequest(RequestContext
requestContext) {
 ResponseContext responseContext = null;
 try {
      String commandName =
requestContext.getCommandName();
```


Capa presentación
Controlador de aplicación

```
CommandFactory commandFactory =
CommandFactory.getInstance();
Command command =
commandFactory.getCommand(commandName);
CommandProcessor commandProcessor = new
CommandProcessor();
responseContext = commandProcessor.invoke(command,
            requestContext);
      } catch (java.lang.InstantiationException e) {
      } catch (java.lang.IllegalAccessException e) {
      }
      return responseContext; }
}
```


Capa de presentación
Contexto

- Propósito
  - Se desea evitar utilizar información del sistema específica del protocolo fuera de su contexto relevante
- También conocido como
  - Context

Capa de presentación
Contexto

- Motivación
  - Al controlador frontal le pueden llegar datos de entrada en formato específico de un protocolo concreto (p.e. `HttpServletRequest` o `JTextField`)
  - Incluir datos de este tipo fuera de su contexto relevante puede provocar un fuerte acoplamiento

Capa de presentación
Contexto

– Debe proporcionarse algún mecanismo que elimine este acoplamiento: utilizar un objeto contexto para encapsular estado de un modo independiente del protocolo para que pueda ser compartido por toda la aplicación

Capa de presentación
Contexto

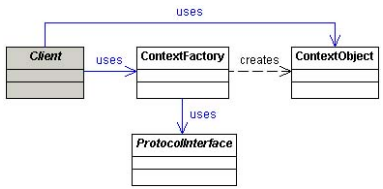
- Contexto
  - Existen componentes y servicios que necisatn acceder a información del sistema
  - Se desea desacoplar componentes de aplicación y servicios específicos del protocolo de la información del sistema
  - Se desea exponer sólo las APIs relevantes dentro de un contexto

Capa de presentación
Contexto

- Solucion
  - Utilizar un objeto contexto para encapsular el estado de una forma independiente del protocolo para ser compartida por toda la aplicación

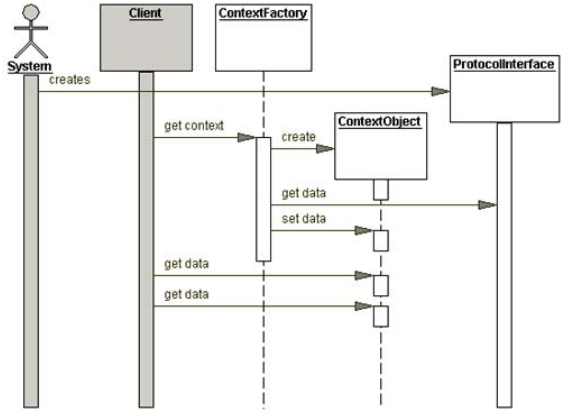
Capa de presentación
Contexto

- Descripción



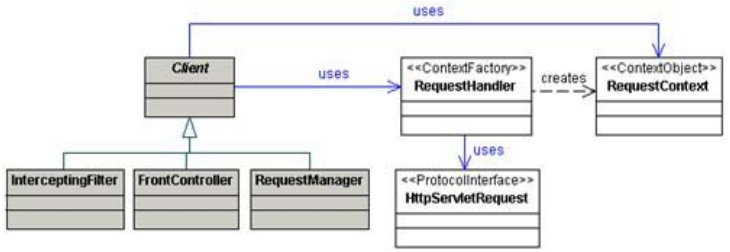
Estructura del patrón contexto

Capa de presentación
Contexto



Interacción en el patrón contexto

Capa de presentación
Contexto



Estructura del request context

Capa de presentación
Contexto

- Consecuencias
  - Ventajas:
    - Mejora reusabilidad y mantenibilidad
    - Mejora las pruebas
    - Reduce las restricciones en la evolución de GUIs
  - Inconvenientes:
    - Reduce el rendimiento

Capa de presentación
Contexto

- Ejemplo

```
public class FrontController extends HttpServlet {

private ApplicationController applicationController;

public void init(ServletConfig servletConfig) throws
ServletException{ super.init(servletConfig);
// Initialize Request Processing(Stateless) Components
applicationController = new
ApplicationControllerImpl();
applicationController.initialize(); }
```

Capa de presentación
Contexto

```
// called from doGet and doPost
protected void process(HttpServletRequest request, 16
HttpServletResponse response) throws
java.io.IOException {

// Create RequestContext based on request type
RequestContextFactory requestContextFactory =
RequestContextFactory.getInstance();
RequestContext requestContext =
requestContextFactory.createRequestContext(request);

// Request Processing
ResponseContext responseContext =
applicationController.handleRequest(requestContext);
```


Capa de presentación
Contexto

```
 // View Management - Navigate and Dispatch to
 //appropriate view

Dispatcher dispatcher = new Dispatcher(request,
response);
responseContext.setDispatcher(dispatcher);
applicationController.handleResponse(requestContext,
responseContext); }

 . . . }
```


Capa de presentación
Contexto

```
public class ApplicationControllerImpl implements
ApplicationController {
public void initialize() {
commandMapper = CommandMapper.getInstance();  }

public ResponseContext handleRequest(RequestContext
requestContext)
{ ResponseContext responseContext = null;
  try { // validate request parameters
           requestContext.validate();
     // Translate command name into Command abstraction
String commandName = requestContext.getCommandName();
```


Capa de presentación
Contexto

```
  Command command =
                commandMapper.getCommand(commandName);
// Invoke Command
  responseContext = command.execute(requestContext);

// Identify View Name
   CommandMap mapEntry =
        commandMapper.getCommandMap(commandName);
        String viewName = mapEntry.getViewName();
        responseContext.setLogicalViewName(viewName);
} catch (ValidatorException e1) { // Handle Exception
}
return responseContext; } . . .
```

Capa de presentación
Contexto

```
// POJO ContextObject Factory
public class RequestContextFactory {
public RequestContext createRequestContext(ServletRequest
request) {
RequestContext requestContext = null;
try { // Identify command string from request object
       String commandId = getCommandId(request);
       // Identify POJO RequestContext Class for the
       //given Command, using CommandMap
       CommandMapper commandMapper =
       CommandMapper.getInstance();
       CommandMap mapEntry =
       commandMapper.getCommandMap(commandId);
```


Capa de presentación
Contexto

```
       Class requestContextClass =
       mapEntry.getContextObjectClass();
       // Instantiate POJO
       requestContext = (RequestContext)
       requestContextClass.newInstance();
       // Set Protocol-specific Request object
       requestContext.initialize(request);  }
catch(java.lang.InstantiationException e)
{ // Handle Exception  }
catch(java.lang.IllegalAccessException e) { }
return requestContext; }
```


Capa de presentación
Contexto

```
private String getCommandId(ServletRequest request) {
 String commandId = null;
 if ( request instanceof HttpServletRequest) {  String
pathInfo = ((HttpServletRequest)request).getPathInfo();
commandId = pathInfo.substring(1);
// skip the leading '/'  }
return commandId; }  }
```


Capa de presentación
Ayudante de vista

- Propósito
  - Se desea separar la lógica de su lógica de procesamiento
- También conocido como
  - View helper

Capa de presentación
Ayudante de vista

- Motivación
  - Sabemos que se debe separar procesamiento de negocio de presentación
  - El controlador frontal ayuda a este fin
  - Sin embargo, procesar una petición involucra dos tipos de actividades:
    - Manejado de la petición
    - Procesamiento en vista

Capa de presentación
Ayudante de vista

  - El procesamiento en vista incluye:
    - Preparación de la vista:
      - Poblar objetos del modelo
      - Identificar y/o generar componentes vista
      - Suelen participar transfers y/u objetos del negocio
    - Creación de la vista:
      - Tranformar el estado del modelo en contenido dinámico que es incluido dentro del componente vista
  - Precisamente un ayudante de vista se encarga de la creación de la vista

Capa de presentación
Ayudante de vista

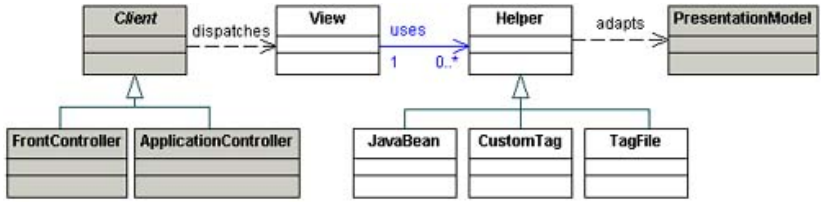
- Contexto
  - Se desea utilizar vistas basadas en plantillas, tales como JSPs
  - No se desea incluir lógica de programa en la vista
  - Se desea separar la lógica de programación de la vista para facilitar la división de tareas entre desarrolladores software y diseñadores de páginas web

Capa de presentación
Ayudante de vista

- Solución
  - Utilizar vistas para encapsular código de formateado y ayudantes para encapsular lógica de procesamiento de vista
  - Una vista delega su responsabilidadesd de procesamiento a sus clases ayudantes
  - Los ayudantes llevan a cabo procesamiento relacionada con la lógica de formateo, como generar una tabla HTML

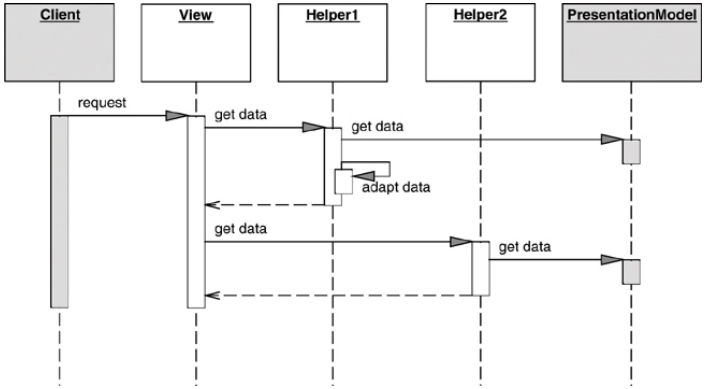
Capa de presentación
Ayudante de vista

- Descripción



Estructura del patrón ayudante de vista

Capa de presentación
Ayudante de vista



Interacción en el patrón ayudante de vista

Capa de presentación
Ayudante de vista

- Consecuencias
  - Ventajas
    - Mejora la partición, reutilización y mantenibilidad
    - Mejora la separación de roles
    - Facilita la prueba
  - Inconvenientes
    - Que el ayudante se convierta en un scriplet

Capa de presentación
Ayudante de vista

- Ejemplo

```
<jsp:useBean id="welcomeHelper" scope="request"
class="corepatterns.util.WelcomeHelper" />
<HTML>
<BODY bgcolor="FFFFFF">
<c:if test = "${welcomeHelper.nameExists == true}">
<center><H3> Welcome
<b><c:out value='${welcomeHelper.name}'/> </b><br><br>
</H3>
</center>
</c:if>
<H4><center>Glad you are visiting our site!</center></H4>
</BODY></HTML>
```

Capa de presentación
Vista compuesta

- Propósito
  - Se desea construir una vista de partes de componentes atómicos y modulares que son combinados para crear un todo compuesto, a pesar de manejar el contenido y composición de forma independiente
- También conocido como
  - Composite view

Capa de presentación
Vista compuesta

- Motivación
  - El desarrollo y mantenimiento de vistas dinámicas es complejo, al incluir aspectos de contenidos y distribución comunes a distintas vistas
  - Cuando se mezclan contenido y distribución es más difícil mantener y extender las vistas
  - La reutilización y modularidad también se resienten al duplicar código común entre vistas

Capa de presentación
Vista compuesta

- Contexto
  - Se desean subvistas comunes, tales como cabeceras, pies y tablas reutilizadas en multiples vistas, las cuales pueden aparecer en distintas localizaciones dentro de cada distribución de página
  - Se tiene contenidos en subvistas que podría cambiar frecuentemente, o depender de controles de acceso en función de roles

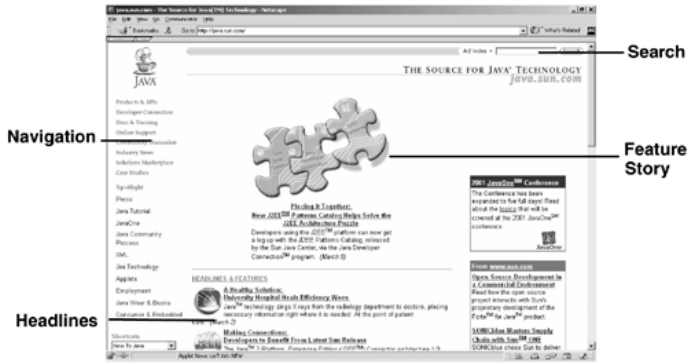
Capa de presentación
Vista compuesta

  - Se desea evitar incluir directamente y duplicar subvistas en multiples vistas, lo que haría cambios en la distribución difíciles de manejar y mantener

Capa de presentación
Vista compuesta

- Solución
  - Utilizar vistas compuestas que estan compuestas de múltiples subvistas atómicas
  - Cada subvista de la plantilla total puede ser incluida dinámicamente en el total, y la distribución de la página puede ser manejada independientemente del contenido

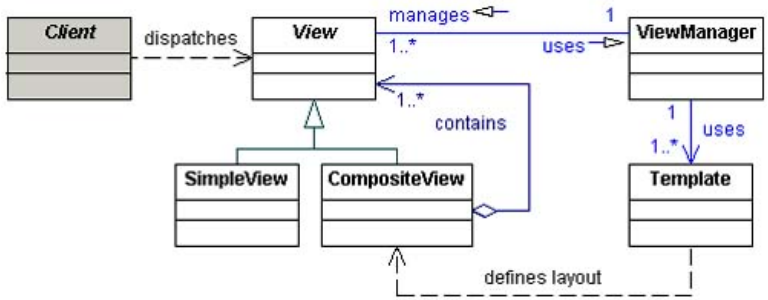
Capa de presentación
Vista compuesta



Página modular formada por distintas regiones

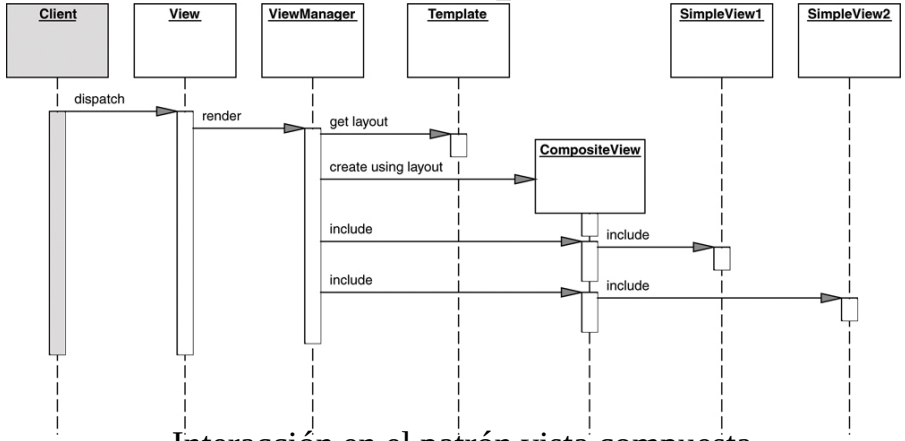
Capa de presentación
Vista compuesta

- Descripción



Estructura del patrón vista compuesta

Capa de presentación
Vista compuesta



Interacción en el patrón vista compuesta

Capa de presentación
Vista compuesta

- Consecuencias
  - Ventajas
    - Mejora la modularidad y reutilizabilidad
    - Añade control basado en el rol o en la política
    - Mejora la mantenibilidad
  - Inconvenientes
    - Reduce la mantenibilidad
    - Reduce el rendimiento

Capa de presentación
Vista compuesta

- Ejemplo

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib uri="/web-INF/corej2eetaglibrary.tld" prefix="cjp" %>
<jsp:useBean id="contentFeeder"
class="corepatterns.compositeview.javabean.ContentFeeder"
scope="request" />
<table valign="top" cellpadding="30%" width="100%">
       <cjp:personalizer interest='global'>
       <tr>
              <td><B><c:out value="${contentFeeder.worldNews}"/>
                  </B></td>
       </tr>
</cjp:personalizer>
```


Capa de presentación
Vista compuesta

```
<cjp:personalizer interest='technology'>
<tr>
  <td><U><c:out value="${contentFeeder.technologyNews}"/>
     </U></td> 21
</tr></cjp:personalizer>

<cjp:personalizer interest='weird'>
<tr>
  <td><I><c:out value="${contentFeeder.weirdNews}"/>
      </I></td> </tr>
</cjp:personalizer>
```


Capa de presentación
Vista compuesta

```
<cjp:personalizer interest='astronomy'>
<tr><td><c:out value="${contentFeeder.astronomyNews}"/>
    </td></tr>
</cjp:personalizer>
</table>
```

Capa de presentación
Vista compuesta

```
<html>
<body>
<jsp:include page="/jsp/CompositeView/javabean/banner.seg"
flush="true"/>
  <table width="100%">

      <tr align="left" valign="middle">
      <td width="20%">
      <jsp:include
page="/jsp/CompositeView/javabean/ProfilePane.jsp" flush="true"/>
      </td>
```


Capa de presentación
Vista compuesta

```
      <td width="70%" align="center">
      <jsp:include
page="/jsp/CompositeView/javabean/mainpanel.jsp"
flush="true"/>
      </td></tr>
</table>

<jsp:include
page="/jsp/CompositeView/javabean/footer.seg"
flush="true"/>
</body>
</html>
```


Capa de presentación
Service to worker

- Propósito
  - Se desea llevar a cabo el manejo de peticiones y la invocación de lógica del negocio antes de pasar el control a la vista
- También conocido como
  - Servicio al trabajador

Capa de presentación
Service to worker

- Motivación
  - Articula en un solo patrón diversos patrones de presentación:
    - Controlador frontal
    - Controlador de aplicación
    - Ayudante de vista

Capa de presentación
Service to worker

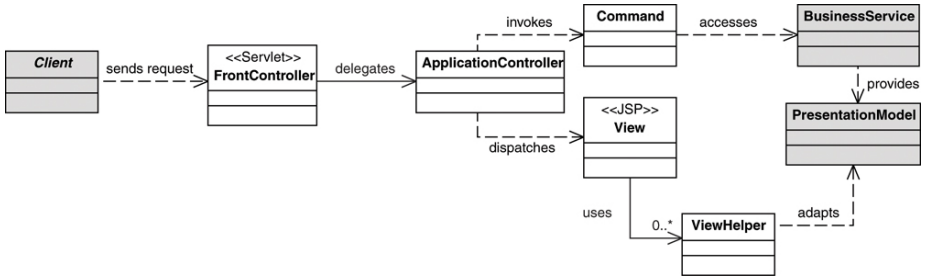
- Contexto
  - Se desea ejecutar lógica de negocio específica para servir una petición y obtener contenido que será utilizado para generar una respuesta dinámica
  - Existe una selección de vistas que puede depender de las respuestas de las invocaciones a los servicios de negocio

Capa de presentación
Service to worker

- Solución
  - Utilizar servicio a los trabajadores para centralizar el control y el manejo de peticiones para obtener un modelo para presentación antes de devolver el control a la vista
  - La vista genera una respuesta dinámica basada en el modelo de presentación

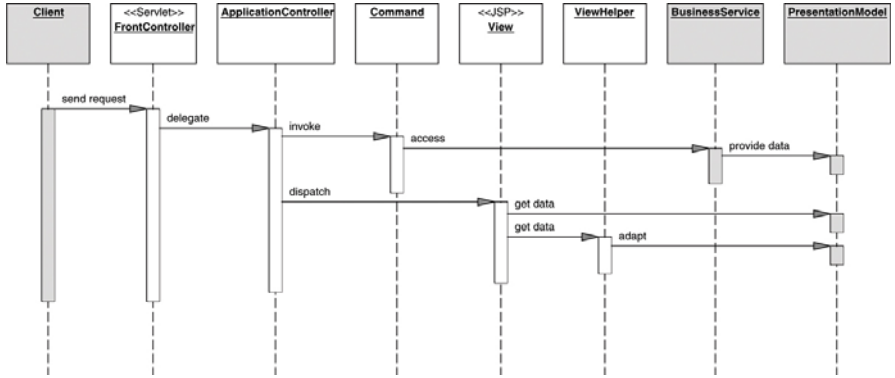
Capa de presentación
Service to worker

- Descripción



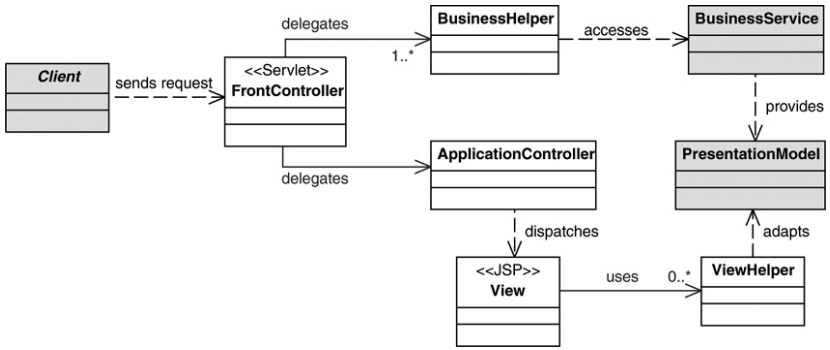
Estructura del patrón service to worker

Capa de presentación
Service to worker



Interacción en el patrón service to worker

Capa de presentación
Service to worker



Estructura alternativa del patrón service to worker

Capa de presentación
Service to worker

- Consecuencias
  - Ventajas
    - Centraliza el control y mejora la modularidad, reusabilidad y mantenibilidad
    - Mejora la separación de papeles
  - Inconvenientes
    - Complejidad

Capa de presentación
Dispatcher view

- Propósito
  - Se desea que una vista maneje una petición y genere una respuesta, gestionando una cantidad limitada de procesamiento de negocio
- También conocido como
  - Vista despachadora

Capa de presentación
Dispatcher view

- Motivación
  - En ciertos casos hay poco, o nulo, procesamiento de negocio llevado a cabo antes de generar la vista
  - Este es el caso cuando la vista es estática o generada de un modelo de presentación existente
  - La vista necesita servicios limitados de negocio o acceso a datos

Capa de presentación
Dispatcher view

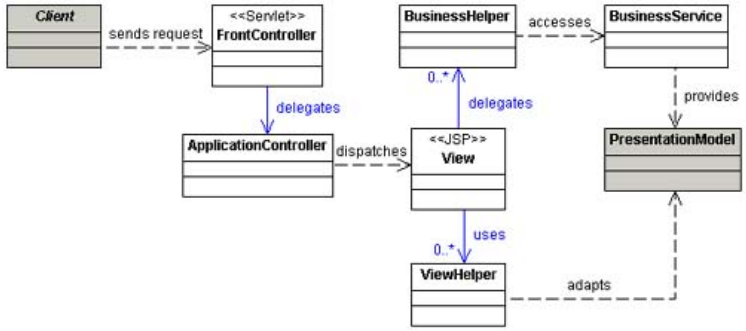
- Contexto
  - Hay vistas estáticas
  - Hay vistas generadas a partir de un modelo de presentación existente
  - Hay vistas independientes de cualquier respuesta de servicio de negocio
  - Hay procesamiento de negocio limitado

Capa de presentación
Dispatcher view

- Solución
  - Utilizar dispatcher view con vistas como el punto de acceso inicial para peticiones. Hay dos opciones comunes:
    - La respuesta es totalmente estática (p.e. HTML)
    - La respuesta es dinámica, pero generada de un modelo de presentación existente (p.e. un campo almacenado en sesión reutilizable por distintas vistas)

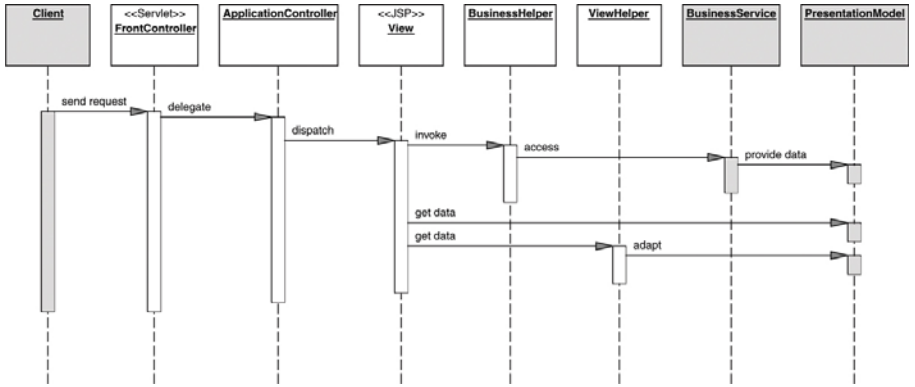
Capa de presentación
Dispatcher view

- Descripción



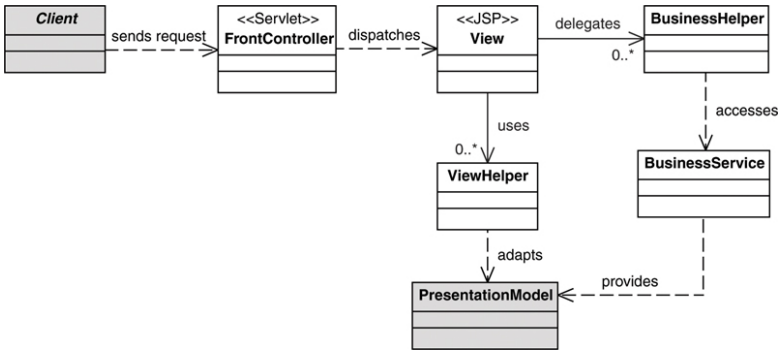
Estructura del patrón dispatcher view

Capa de presentación
Dispatcher view



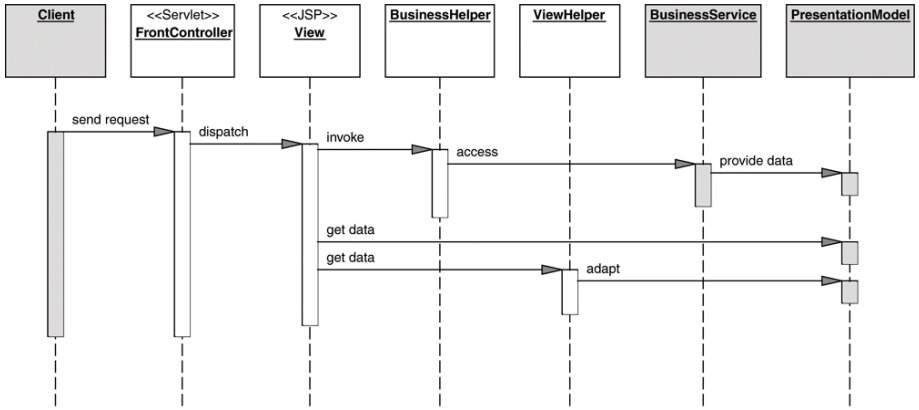
Interacción en el patrón dispatcher view

Capa de presentación
Dispatcher view



Estructura alternativa del patrón dispatcher view

Capa de presentación
Dispatcher view



Interacción alternativa en el patrón dispatcher view

Capa de presentación
Dispatcher view

- Consecuencias
  - Ventajas
    - Aprovechamiento de marcos y librerías
  - Inconvenientes
    - Introduce una potencial falta de separación de la vista del modelo y la lógica de control

Capa de negocio
Delegado del negocio

- Propósito
  - Evita que los clientes tengan que tratar con detalles de acceso a componentes distribuidos en una aplicación multicapa
- También conocido como:
  - *Business delegate*

Capa de negocio
Delegado del negocio

- Motivación
  - Cuando los clientes interactúan con servicios de negocio remotos, pueden ocurrir diversos problemas:
    - Acoplamiento del cliente con el servicio
    - Rendimiento de la red por múltiples invocaciones
    - Exposición de los detalles de acceso al servicio

Capa de negocio
Delegado del negocio

- Contexto
  - Se desea acceder a componentes remotos de negocio
  - Se desea minimizar el acoplamiento entre los clientes y los componentes remotos
  - Se desea evitar invocaciones remotas innecesarias

Capa de negocio
Delegado del negocio

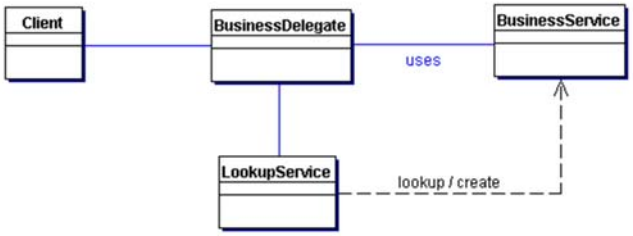
  - Se desea traducir excepciones de red en excepciones de aplicación o de usuarios
  - Se desea ocultar los detalles de la creación, reconfiguración e intentos de invocación de los servicios a los clientes

Capa de negocio
Delegado del negocio

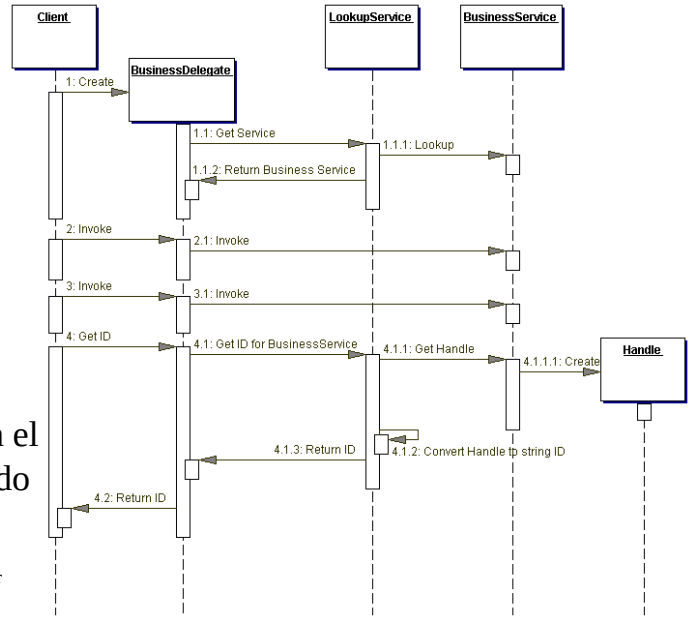
- Solucion
  - Utilizar un delegado del negocio para encapsular el acceso a los servicios de negocio remotos
  - El delegado del negocio oculta los detalles de implementación del servicio de negocio, tales como los mecanismos de búsqueda y acceso

Capa de negocio
Delegado del negocio

- Descripción



Estructura del patrón delegado del negocio



Interacción en el patrón delegado del negocio

Capa de negocio
Delegado del negocio

- Consecuencias
  - Ventajas
    - Reduce el acoplamiento, mejora la modularidad
    - Traduce excepciones de servicios de negocio
    - Mejora la disponibilidad
    - Expone un interface uniforme y más simple a la capa de negocio
    - Mejora el rendimiento

Capa de negocio
Delegado del negocio

- Inconvenientes
  - Introduce un nivel adicional
  - Oculta la localización de los servicios remotos

Capa de negocio
Delegado del negocio

- Código de ejemplo

```
public class StockListDelegate {
  private StockList stockList;

  //accede al objeto fachada de la aplicación
  private StockListDelegate() throws StockListException
  {
      try { InitialContext ctx= new InitialContext();
            stockList=
   (StockList) ctx.lookup(StockList.class.getName());
           } catch(Exception e) {
             throw
            new StockListException(e.getMessage());
            }
  }
```


Capa de negocio
Delegado del negocio

```
.....
//stock es un objeto transferencia para las acciones
public void addStock(StockTO stock) throws
  StockListException {

  //delega en la fachada
  try { stockList.add(stock); }
  catch (Exception re) {
      throw new StockListException (re.getMessage());
      }
}
.....
```


Capa de negocio
Delegado del negocio

```
.....
//el delegado en un singleton
public static StockListDelegate getInstance()
  throws StockListException {

  if (stockListDelegate == null)
      stockListDelegate= new StockListDelegate();

  return stockListDelegate;
}
}
```


Capa de negocio
Delegado del negocio

```
//interfaz remoto de la fachada
@Remote
public interface StockList {

  public List getStockRatings();
  public List getAllAnalysts();
  public List getUnratedStocks();
  public void addStockRating(StockTO stockTO);
  public void addAnalystAnalystTO analystTO);
  public void addStock(StockTo stockTO);
}
```

Capa de negocio
Delegado del negocio

```
//implementación de la fachada como EJB de sesión
//sin estado
@Stateless
public class StockListBean implements StockList
{
   ……..

}
```


Capa de negocio
Localizador de servicio

- Problema
  - Se desea localizar de manera transparente componentes de negocio y servicios de una manera uniforme
- También conocido como
  - Service locator

Capa de negocio
Localizador de servicio

- Motivación
  - Las aplicaciones necesitan localizar y acceder a componentes de la capa de negocios y servicios
  - Esta es una tarea compleja

Capa de negocio
Localizador de servicio

- Contexto
  - Se desea utilizar el API JNDI* par localizar componentes de negocio (p.e. EJBs, componentes JMS, datasources)
  - Se desea centralizar y reutilizar la implementación de mecanismos de búsqueda en los clientes

*Java Naming and Directory Interface

Capa de negocio
Localizador de servicio

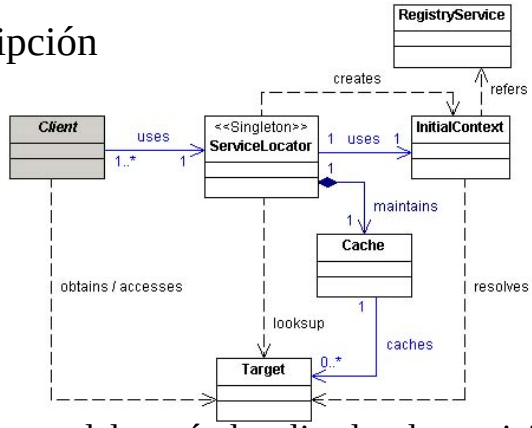
- Se desea encapsular dependencias de implementaciones para las implementaciones y ocultar la dependencia y complejidad a los clientes
- Se desea evitar la sobrecarga de rendimiento vinculada a la creación de contextos iniciales y servicios de búsqueda

Capa de negocio
Localizador de servicio

- Solucion
  - Utilizar un localizador de servicio para implementar y encapsular el servicio y componente de búsqueda
  - Un localizador de servicio oculta los detalles de implementación del mecanismo de búsqueda y encapsula las dependencias relacionadas

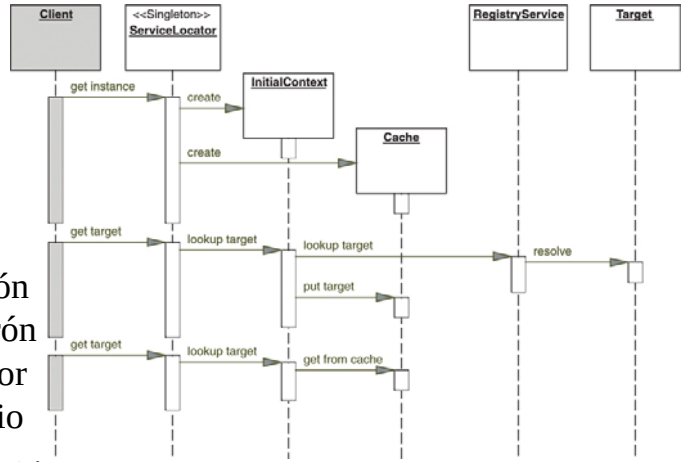
Capa de negocio
Localizador de servicio

- Descripción



Estructura del patrón localizador de servicio

Capa de negocio
Localizador de servicio



Interacción en el patrón localizador de servicio

Capa de negocio
Localizador de servicio

- Consecuencias
  - Ventajas
    - Abstrae la complejidad
    - Proporciona a los clientes un acceso uniforme a los servicios
    - Facilita añadir componentes de negocio EJBs
    - Mejora el rendimiento de red
    - Mejora el rendimiento del cliente al *cachear*

Capa de negocio
Localizador de servicio

- Ejemplo

```
package com.corej2eepatterns.servicelocator;
// imports

public class ServiceLocator {
 . . .

public DataSource getDataSource(String dataSourceName)
throws ServiceLocatorException {
DataSource dataSource = null;
```


Capa de negocio
Localizador de servicio

```
try {
 if (cache.containsKey(dataSourceName)) {
             dataSource = (DataSource)
                          cache.get(dataSourceName);
} else {dataSource = (DataSource)
                     initialContext.lookup(dataSourceName);
                     cache.put(dataSourceName, dataSource );
       }
} catch (NamingException nex) {
       throw new ServiceLocatorException(nex); }
catch (Exception ex) { throw new ServiceLocatorException(ex);
}
return dataSource;}
. . .}
```


Capa de negocio
Fachada de sesión

- Propósito
  - Se desea exponer componentes de negocio y servicios a clientes remotos
- También conocido como
  - Session Façade

Capa de negocio
Fachada de sesión

- Motivación
  - La fachada de sesión soluciona dos problemas:
    - Controlar el acceso de clientes a objetos del negocio
    - Limitar el tráfico de red entre clientes remotos y componentes de negocio y servicios de grano fino

Capa de negocio
Fachada de sesión

- Contexto
  - Se desea evitar dar acceso directo a los clientes a componentes de la capa de negocio, para evitar acoplamientos fuertes
  - Se desea proporcionar una capa de acceso remoto para los objetos del negocio y otros componentes de la capa de negocio
  - Se desea agregar y exponer los servicios de aplicación y otros servicios a clientes remotos

Capa de negocio
Fachada de sesión

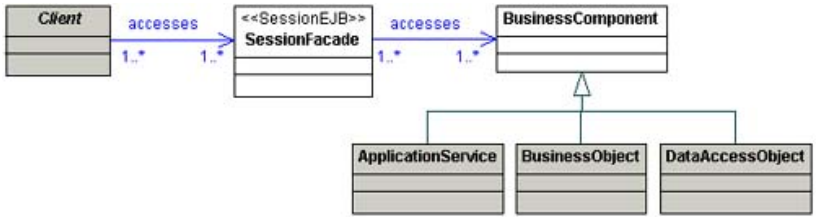
  - Se desea centralizar y agregar toda la lógica de negocio que necesita ser expuesta a clientes remotos
  - Se desea ocultar interacciones e interdependencias complejas entre componentes de negocio y servicios para mejorar la manejabilidad, centralizar la lógica, incrementar la flexibilidad y mejorar la capacidad para gestionar los cambios

Capa de negocio
Fachada de sesión

- Solución
  - Utilizar una fachada de sesión para encapsular los componentes de la capa de negocio y exponer un servicio de grano grueso a clientes remotos
  - Los clientes acceden a una fachada de sesión, en vez de acceder directamente a los componentes de negocio

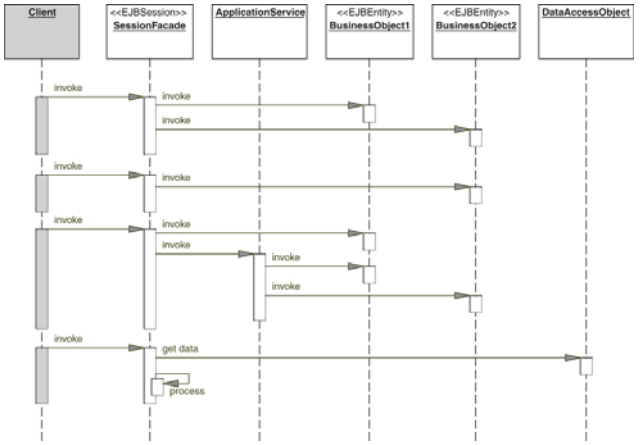
Capa de negocio
Fachada de sesión

- Descripción



Estructura del patrón fachada de sesión

Capa de negocio
Fachada de sesión



Interacción en el patrón fachada de sesión

Capa de negocio
Fachada de sesión

- Consecuencias
  - Ventajas
    - Introduce una capa que proporciona servicios a clientes remotos
    - Expone un interfaz uniforme de grano grueso
    - Reduce el acoplamiento entre capas
    - Promueve la división en capa, incrementa la flexibilidad y mantenibilidad
    - Reduce la complejidad

Capa de negocio
Fachada de sesión

- Mejora el rendimiento, reduce los métodos remotos de grano fino
- Centraliza la gestión de seguridad
- Centraliza el control de transacción
- Expone menos interfaces remotos a los clientes

Capa de negocio
Fachada de sesión

- Ejemplo*

```
package org.soademo.customerservice.business;

import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import
org.soademo.customerservice.persistence.Customer;

@Stateless(name = "CustomerService")
public class CustomerServiceBean implements
CustomerService, CustomerServiceLocal {
```

*http://sqltech.cl/doc/oas10gR31/core.1013/b28764/ejb004.htm

Capa de negocio
Fachada de sesión

```
@PersistenceContext(unitName = "customerServiceUnit" )
    private EntityManager em;
    public CustomerServiceBean() { }

    public Object mergeEntity(Object entity) {
        return em.merge(entity);
    }

    public Object persistEntity(Object entity) {
        em.persist(entity);
        return entity;
    }
```


Capa de negocio
Fachada de sesión

```
 public Object refreshEntity(Object entity) {
        em.refresh(entity);
        return entity;  }

    public void removeEntity(Object entity) {
        em.remove(em.merge(entity));
    }

    /** <code>select object(cust) from Customer cust
where cust.custid = :custid</code> */
```


Capa de negocio
Fachada de sesión

```
 public Customer queryCustomerFindCustomerById(String
custid) {
        return
(Customer)em.createNamedQuery("Customer.findCustomerByI
d").setParameter("custid",

custid).getSingleResult();
    }

    public String getCustomerStatus(String CustomerID)
{
        return
findCustomerById(CustomerID).getStatus();
```

Capa de negocio
Fachada de sesión

```
public String addNewCustomer(Customer customer) {
        em.persist(customer);
        return "New customer added sucessfully to customer
database";
    }
     public Customer findCustomerByEmail(String email,
String password) {
        return
(Customer)em.createNamedQuery("Customer.findCustomerByEmail
").setParameter("email", email).setParameter("password",
password).getSingleResult();
    }
 }
```


Capa de negocio
Servicio de aplicación

- Propósito
  - Se desea centralizar la lógica de negocio a través de distintos componentes de la capa de negocio y servicios
- También conocido como:
  - Application service

Capa de negocio
Servicio de aplicación

- Motivación
  - Las fachadas no contienen poca o nula lógica de negocio y exponen un simple interfaz de grano grueso
  - Los objetos de negocio representan objetos persistentes
  - Sin embargo existen reglas de negocio que deben ser implementadas en algún objeto

Capa de negocio
Servicio de aplicación

- Contexto
  - Se desea minimizar la lógica de negocio de las fachadas de servicio
  - Existe lógica de negocio actuando sobre distintos objetos de negocio y/o servicios
  - Se desea proporcionar un API de servicio de grano grueso sobre componentes de la capa de negocio y servicios existentes

Capa de negocio
Servicio de aplicación

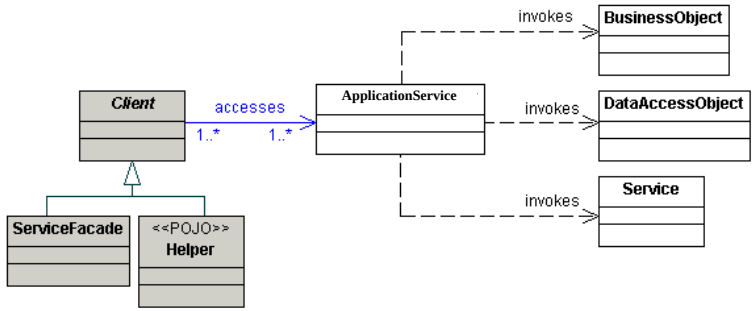
– Se desea encapsular lógica de negocio específica del caso fuera de objetos del negocio individuales

Capa de negocio
Servicio de aplicación

- Solución
  – Utilizar un servicio de aplicación para centralizar y agregar comportamiento para proporcionar una capa de servicio uniforme

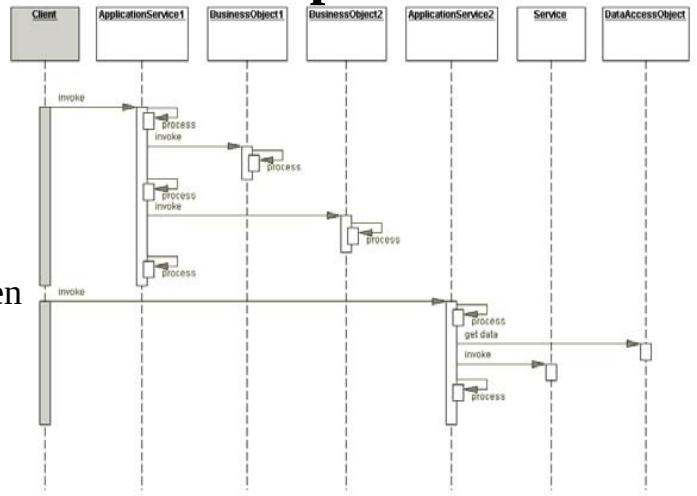
Capa de negocio
Servicio de aplicación

- Estructura



Estructura del patrón servicio de aplicación

Capa de negocio
Servicio de aplicación



Interacción en el patrón servicio de aplicación

Capa de negocio
Servicio de aplicación

- Consecuencias
  - Ventajas
    - Centraliza lógica del negocio y de workflow reutilizable
    - Mejora la reusabilidad de la lógica de negocio.
    - Evita duplicación de código.
    - Simplifica la implementación de fachadas
  - Inconvenientes
    - Introduce un nivel más de indirección

Capa de negocio
Servicio de aplicación

- Ejemplo

```
public interface Biblioteca {
   public Integer insertaUsuario(TUsuario usuario);
   public Boolean daDeBajaUsuario(Integer id);
   public TUsuario obtenUsuario(Integer id);
   public Integer insertaPublicacion(TPublicacion
   publicacion);
   public Boolean daDeBajaPublicacion(Integer id);
   public TPublicacion obtenPublicacion(Integer id);
   public TPrestamo prestamo(TPrestamo tPrestamo);
   public Boolean devolucion(Integer ejemplar); }
```


Capa de negocio
Servicio de aplicación

```
public class BibliotecaImp implements Biblioteca {
……

   public TPrestamo prestamo(TPrestamo tPrestamo)
   {   //nótese que no se ha hecho explícito
       //el acceso a estos servicios por parte de la
       //Biblioteca
       serviciosPrestamo.prestamo(tPrestamo);
   }
……
}
```


Capa de negocio
Servicio de aplicación

```
package logica.serviciosPrestamo;

import transferenciaCliente.prestamo.TPrestamo;


public interface ServiciosPrestamo {
   public TPrestamo prestamo(TPrestamo tPrestamo);

   public Boolean devolucion(Integer ejemplar);
}
```

Capa de negocio
Servicio de aplicación

```
package logica.serviciosPrestamo;
……….

public class ServiciosPrestamoImp implements
   Servicios Prestamo {
   public TPrestamo prestamo(TPrestamo tPrestamo)
   { …… }

   public Boolean devolucion(Integer ejemplar)
   { …… }


}
```


Capa de negocio
Objeto del negocio

- Propósito
  - Se tiene un modelo del dominio conceptual con lógica del negocio y relaciones
- También conocido como:
  - Business object

Capa de negocio
Objeto del negocio

- Motivación
  - Cuando la lógica del negocio es poca o inexistente, las aplicaciones pueden permitir a los clientes acceder directamente a la capa de datos.
  - Así, un componente de la capa de negocio (e.g. `ServiciosUsuarioImp`) podría acceder directamente a un DAO.

Capa de negocio
Objeto del negocio

  - Esto lleva a un tipo de programación procedural, no orientada a objetos
  - Si embargo, si el modelo conceptual es complejo, esta aproximación no es la más adecuada

Capa de negocio
Objeto del negocio

- Contexto
  - Se tiene un modelo conceptual conteniendo objetos del negocio interrelacionados y compuestos
  - Se tiene un modelo conceptual con lógica de negocio, reglas de validación y reglas de negocio sofisticadas

Capa de negocio
Objeto del negocio

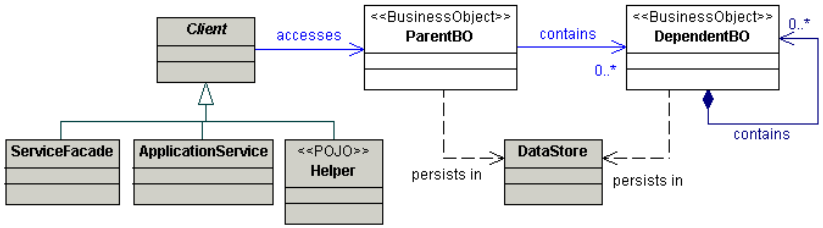
  - Se desea separar el estado del negocio y el comportamiento relacionado del resto de la aplicación, mejorando la cohesión y reusabilidad
  - Se desea centralizar lo lógica del negocio y el estado en una aplicación
  - Se desea incrementar la reusabilidad de la lógica de negocio y evitar la duplicación de código

Capa de negocio
Objeto del negocio

- Solución
  - Utilizar objetos del negocio para separar datos del negocio y lógica utilizando un modelo de objetos
  - Los objetos del negocio encapsula y manejan:
    - Datos del negocio
    - Comportamiento
    - Persistencia

Capa de negocio
Objeto del negocio

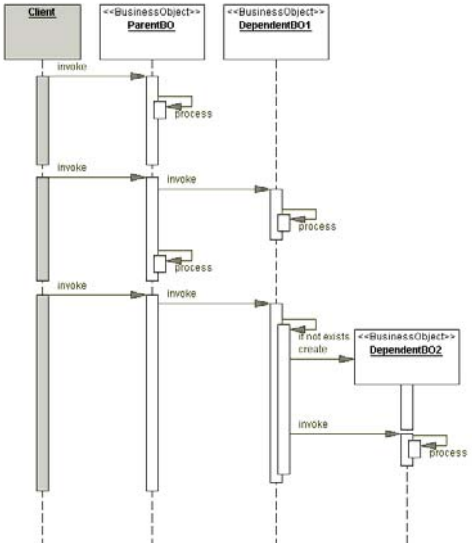
- Descripción

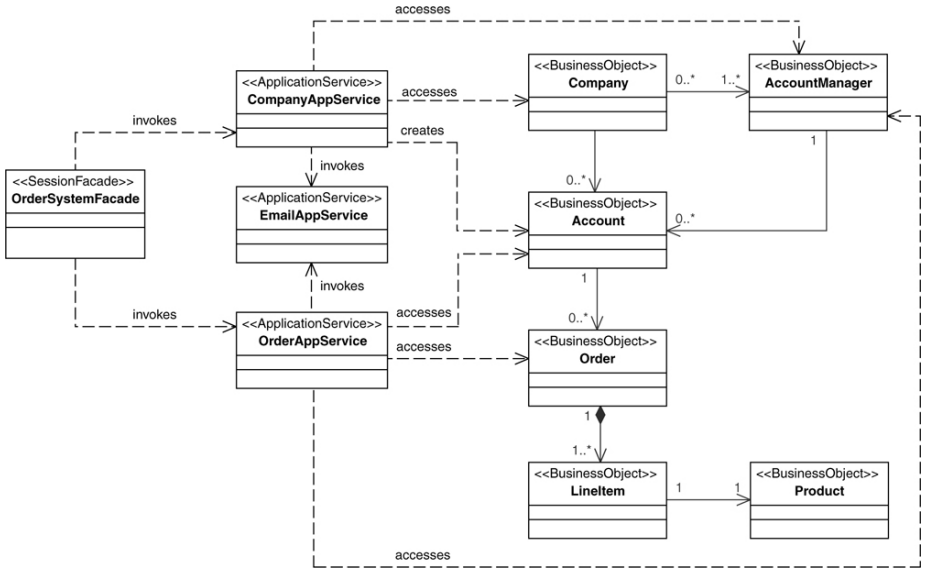


Estructura del patrón objeto del negocio

Capa de negocio

Interacción en el patrón objeto del negocio





Relación entre objetos del negocio y servicios de aplicación

Capa de negocio
Objeto del negocio

- Consecuencias
  - Ventajas
    - Promueve una aproximación orientada a objetos en la implementación del modelo del negocio.
    - Centraliza el comportamiento y estado del negocio, promoviendo la reutilizabilidad.
    - Evita la duplicación de código y mejora su mantenibilidad

Capa de negocio
Objeto del negocio

  - Separa la lógica de persistencia de la lógica de negocio
  - Promueve una arquitectura orientada a servicos
- Inconvenientes
  - La implementación POJO puede promover datos caducados y su persistencia es compleja
  - Añade una capa de indirección.
  - Puede producir objetos “inflados” de funcionalidad.

Capa de negocio
Objeto del negocio

- Ejemplo

```
@Stateful
public class DepartmentManagerBean implements
DepartmentManager {
    @PersistenceContext(unitName="EmployeeService")
    EntityManager em;
    int deptId;

    public void init(int deptId) {
        this.deptId = deptId;
    }
```


Capa de negocio
Objeto del negocio

```
public void setName(String name) {
        Department dept = em.find(Department.class,
deptId);
        dept.setName(name);
    }

    public void addEmployee(int empId) {
        Department dept = em.find(Department.class,
deptId);
        Employee emp = em.find(Employee.class, empId);
        dept.getEmployees().add(emp);
        emp.setDepartment(dept);
    }
```


Capa de negocio
Objeto del negocio

```
// ...

    @Remove
    public void finished() {
    }
}
```


Capa de negocio
Entidad compuesta

- Propósito
  - Se dese utilizar entidades JPA* para implementar el modelo conceptual
- También conocido como
  - Composite entity

*Originalmente entity beans, y en general, cualquier marco de persistencia

Capa de negocio
Entidad compuesta

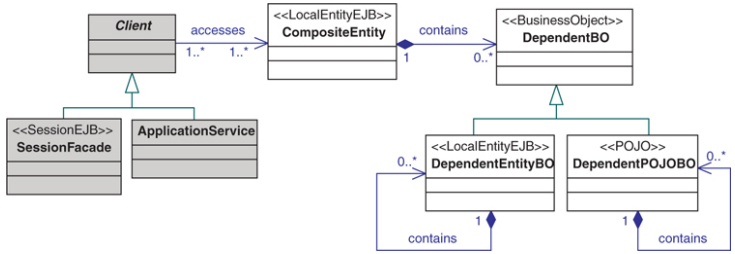
- Motivación
  - Se desea implementar eficientemente relaciones padre-hijo cuando se implementan objetos del negocio como entidades JPA
  - Se desea encapsular el diseño físico de la base de datos de los clientes

Capa de negocio
Entidad compuesta

- Solución
  - Utilizar una entidad compuesta para implementar objetos del negocio persistentes como entidades JPA
  - La entidad compuesta agrega un conjunto de objetos del negocio relacionados en entidades JPA de grano grueso

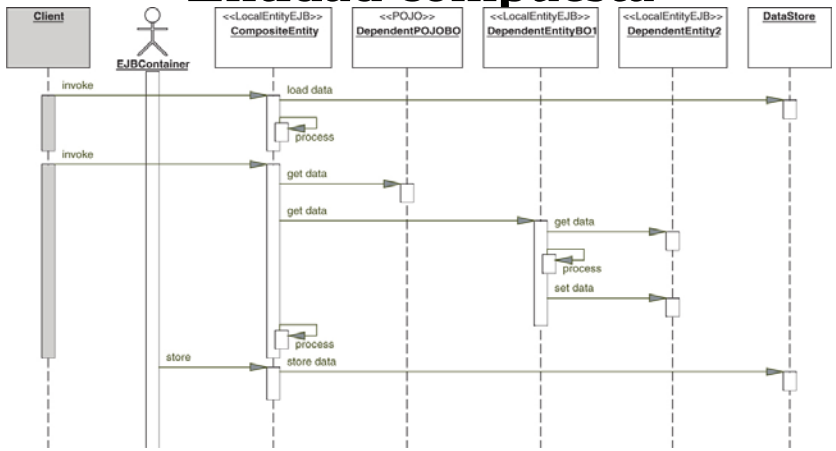
Capa de negocio
Entidad compuesta

- Descripción



Estructura del patrón entidad compuesta

Capa de negocio
Entidad compuesta



Interacción en el patrón entidad compuesta

Capa de negocio
Entidad compuesta

- Consecuencias
  - Ventajas
    - Reduce la dependencia del esquema de la base de datos
    - Incrementa la granularidad de los objetos

Capa de negocio
Entidad compuesta

- Ejemplo*

```
@Entity(name = "CUSTOMER")
public class Customer {
	@Id //signifies the primary key
	@Column(name = "CUST_ID", nullable = false)
	@GeneratedValue(strategy = GenerationType.AUTO)
	private long custId;

	@Column(name = "FIRST_NAME", length = 50)
	private String firstName;
```

*http://www.javaworld.com/javaworld/jw-01-2008/jw-01-jpa2.html

Capa de negocio
Entidad compuesta

```
@Column(name = "LAST_NAME", nullable = false,length = 50)
	private String lastName;

	@Column(name = "STREET")
	private String street;

@OneToMany(mappedBy="customer",targetEntity=Order.class,
	fetch=FetchType.EAGER)
	private Collection orders;
	.........................
	// The other attributes and getters and setters goes
here
}
```


Capa de negocio
Entidad compuesta

```
........
EntityManager em =
entityManagerFactory.createEntityManager();
Customer customer = em.find(Customer.class, 100);
System.out.println("Order details for customer 100 : "
+ customer.getOrders());
em.close();
entityManagerFactory.close();
.........
```

Capa de negocio
Patrón transferencia

- Propósito
  - Independizar el intercambio de datos entre capas
- También conocido como
  - Transfer

Capa de negocio
Patrón transferencia

- Motivación
  - Los datos tienen que fluir entre la capa de negocios y las otras capas
  - Dicho flujo debería ser independiente de los mecanismos de representación de datos de cada capa
  - Si el mecanismo de persistencia lo soporta se podrían utilizar objetos del negocio desvinculado del gestor de persistencia

Capa de negocio
Patrón transferencia

- Además, soporte o no soporte la desvinculación, tenemos el problema adicional de la carga dinámica de referencias

Capa de negocio
Patrón transferencia

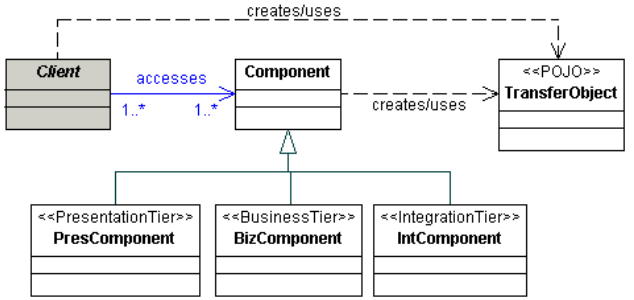
- Contexto
  - Hay clientes que necesitan acceder a componentes en otras capas para recuperar y actualizar datos
  - Los datos deberían ser representaciones orientadas a objetos que ocultasen los mecanismos de representación utilizados en cada capa

Capa de negocio
Patrón transferencia

- Solución
  - Utilizar un objeto transferencia para mover multiples elementos de datos entre capas
  - Al ser un mecanismo de comunicación entre capas, los transfers son objetos serializables

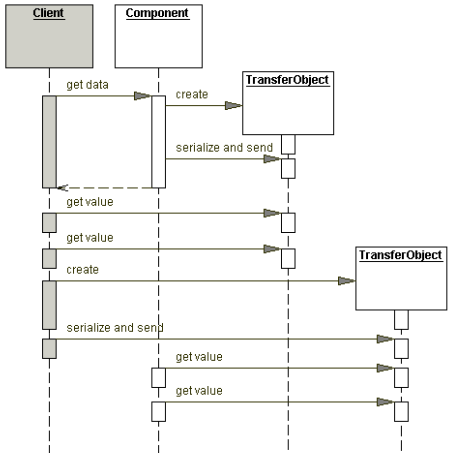
Capa de negocio
Patrón transferencia

- Descripción



Estructura del patrón transferencia

Capa de negocio
Patrón transferencia



Interacción en el patrón transferencia

Capa de negocio
Patrón transferencia

- Consecuencias
  - Ventajas
    - Ayuda a independizar capas
  - Inconvenientes
    - Introduce objetos con datos no actualizados

Capa de negocio
Patrón transferencia

- Código de ejemplo

```
public TransferLibro {
//atributos de libro
public String autor;
……
//accesores y mutadores de libro
public String getAutor()
{ return autor; }
……
public void setAutor(String autor)
{ this.autor= autor; }
……
```


Capa de negocio
Patrón transferencia

```
public DAOEjemplaresImp implements DAOEjemplares {

  public TransferLibro obtenerLibro(String id)
  {
      //código acceso a la base de datos

      TransferLibro libro= new TranserLibro(autor, …);

      return libro;
  }
  ……
}
```


Capa de negocio
TOA

- Propósito
  - Se desea obtener un modelo de aplicación que agregue objetos transferencia de distintos componentes de negocio
- También conocido como
  - Transfer Object Assembler
  - Ensamblador de objetos transferencia

Capa de negocio
TOA

- Motivación
  - Los clientes de aplicación necesitan obtener datos de negocio o un *modelo de aplicación* de la capa de negocio, bien para presentarlo, bien para hacer un procesamiento intermedio
  - Un modelo de aplicación representa datos de negocio encapsulados por componentes de negocio en la capa de negocio

Capa de negocio
TOA

– Cuando los clientes necesitan los datos del modelo de aplicación deben localizar, acceder y obtener diferentes partes del modelo de diferentes fuentes, tales como objetos del negocio, DAOs, servicios de aplicación y otros

– Esta aproximación
  - Acopla a los clientes con diversos componentes de la aplicación
  - Puede provocar la duplicación de código en distintos clientes accediendo a los mismos datos

Capa de negocio
TOA

- Contexto
  – Se desea encapsular lógica de negocio de forma centralizada, evitando su implementación en el cliente
  – Se desea minimizar las llamadas a objetos remotos al construir una representación de datos de modelos de objetos de la capa de negocio

Capa de negocio
TOA

– Se desea crear un modelo complejo para dárselo al cliente en presentación

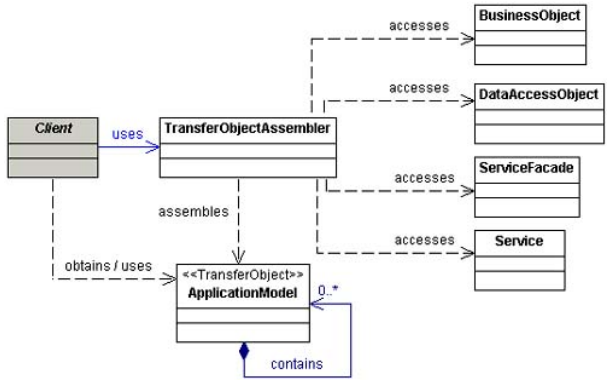
– Se desea que los clientes sean independientes de la complejidad del modelo de implementación, reduciendo el acoplamiento entre el cliente y los componentes de negocio

Capa de negocio
TOA

- Solución
  – Utilizar un TOA para construir un modelo de aplicación como un objeto transferencia compuesto
  – El TOA agrega distintos objetos transferencia provenientes de diversos componentes y lo devuelve al cliente

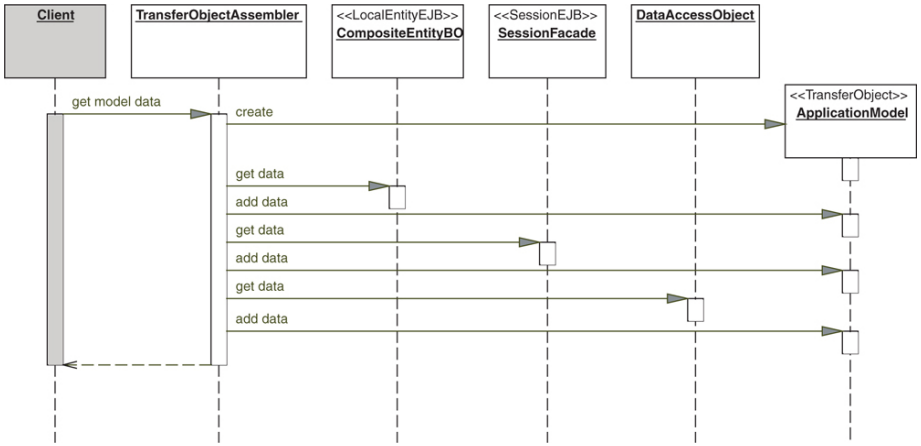
Capa de negocio
TOA

- Descripción



Estructura del patrón TOA

Capa de negocio
TOA



Interacción en el patrón TOA

Capa de negocio
TOA

- Consecuencias
  - Ventajas
    - Separa la lógica de negocio, simplifica la lógica de cliente
    - Reduce el acoplamiento entre clientes y el modelo de aplicación
    - Mejora el rendimiento de la red
    - Mejora el rendimiento del cliente
  - Inconvenientes
    - Puede introducir datos desactualizados

Capa de negocio
TOA

- Ejemplo

```
public class TBestOfShop {
 protected TCustomer bestCustomer;
 protected TProduct bestProduct;
 protected TBrand bestBrand;
……..
}
```

Capa de negocio
TOA

```
public class ShopAS {

 TBestOfShop bestOfShop()
{  ……..
    TCustomer customer= CustomerAS.best();
    TProduct product= ProductAS.best();
    TBrand brand= BrandAS.best;

   return new TBestOfShop(customer, product, brand); }

 …… }
```


Capa de negocio
Manejador de lista de valores

- Propósito
  - Se tiene un cliente remoto que desea iterar sobre una lista de resultados grande
- También conocido como
  - Value list handler

Capa de negocio
Manejador de lista de valores

- Motivación
  - Las aplicaciones suelen tener clientes que llevan a cabo búsquedas
  - A menudo, estas búsquedas las inicia presentación, las lleva a cabo negocio y las visualiza el navegador
  - Si las búsquedas devuelven pocos resultados, no hay problema

Capa de negocio
Manejador de lista de valores

  - Si embargo, si hay muchos resultados es posible que el cliente no sea capaz de manejarlos
  - Además, el cliente muchas veces no necesita revisar todos los resultados, por lo que no es deseable malgastar ancho de banda

Capa de negocio
Manejador de lista de valores

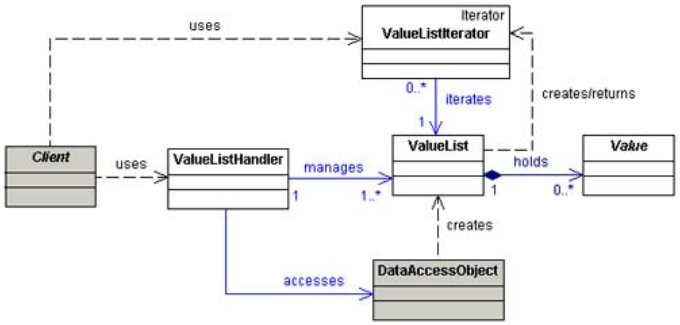
- Contexto
  - Se desea implementar un caso de uso de sólo lectura que no requiere una transacción
  - Se desea proporcionar al cliente con un mecanismo de búsqueda e interacción eficiente sobre un conjunto de resultados grande
  - Se desea mantener los resultados de búsqueda en el servidor

Capa de negocio
Manejador de lista de valores

- Solución
  - Utilizar un manejador de lista de valores para buscar, cachear los resultados, y permitir al cliente recorrer y seleccionar los elementos de los resultados

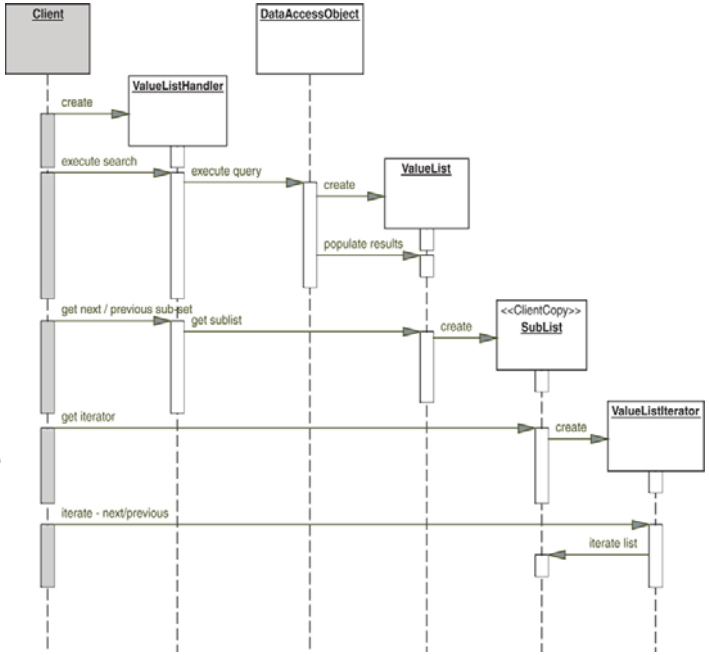
Capa de negocio
Manejador de lista de valores

- Descripción



Estructura del patrón manejador de lista de valores

Interacción en el patrón manejador de lista de valores



Capa de negocio
Manejador de lista de valores

- Consecuencias
  - Ventajas
    - Cachea resultados de búsqueda
    - Proporciona capacidades de búsqueda flexibles
    - Mejora el rendimiento de la red
    - Evita la sobrecarga de transacciones
    - Proporciona partición en capa

Capa de negocio
Manejador de lista de valores

  - Inconvenientes
    - Crear una larga lista de objetos transferencia puede ser costoso
    - Puede introducir datos no actualizados

Capa de negocio
Manejador de lista de valores

- Ejemplo

```
package com.corej2eepatterns.vlh;
// imports
public class ProjectListHandler
extends ValueListHandler {
       private ProjectDAO dao = null;
   . . .
       // Client creates a ProjectTO instance, sets the
       // values to use for search criteria and passes
       // the ProjectTO instance as projectCriteria
       // to the constructor and to setCriteria()
method
```


Capa de negocio
Manejador de lista de valores

```
      public ProjectListHandler()
      throws ProjectException, ListHandlerException {
             try {
                    this.dao =
PSADAOFactory.getProjectDAO();
             } catch (Exception e) {
                    // Handle exception, throw
//ListHandlerException
             }
       }
```

Capa de negocio
Manejador de lista de valores

```
      // executes search. Client can invoke this
      // provided that the search criteria has been
      // properly set. Used to perform search to refresh
      // the list with the latest data.
      public void executeSearch(ProjectTO projectCriteria)
      throws ListHandlerException {
            try {
                  if (projectCriteria == null) {
                        throw new ListHandlerException(
                              "Project Criteria
required...");
                  }
```


Capa de negocio
Manejador de lista de valores

```
List resultsList =

      dao.findProjects(projectCriteria);
                  setList(resultsList);
            } catch (Exception e) {
                  // Handle exception, throw
ListHandlerException
            }
      }
}
```


Capa de negocio
Manejador de lista de valores

```
package com.corej2eepatterns.vlh;
// imports
public abstract class ValueListHandler {
    List valueList;
    ...
    // you only need to implement this method
    public abstract void executeSearch(Object
criteria);
    ...
```


Capa de negocio
Manejador de lista de valores

```
   protected void setList(List valueList) {
       this.valueList = valueList;
   }

   public List getNextElements(
           int startPosition, int endPosition) {
       return valueList.subList(startPosition,
endPosition);
   }
    public List getPreviousElements(
           int startPosition, int endPosition) {
       return valueList.subList(startPosition,
endPosition);
   }
```

Capa de negocio
Manejador de lista de valores

```
public Object getValue(int index) {
        return valueList.get(index);
    }
   public int size() {
      return valueList.size();
    }
    // if using multiple value lists
    // provide methods that accept a list id as an
argument
    // e.g. public List getPreviousElements(int listId,
    // int startPosition, int endPosition), and so on

    ... }
```


Capa de negocio
Manejador de lista de valores

```
package com.corej2eepatterns.vlh;
// imports
public class ProjectsList extends ArrayList {
    ...
    // implement iterator() to return your custom
iterator,
    // if any
    public Iterator iterator() {
        return new ProjectsListIterator(this);
    }
```


Capa de negocio
Manejador de lista de valores

```
 // implement iterator() to return your custom
//iterator, if any
    public ListIterator listIterator() {
        return new ProjectsListIterator(this);
    }
    public List subList(int fromIndex, int toIndex) {
        ProjectsList subList = new ProjectsList();
        for (int index=fromIndex; index<toIndex;
index++) {
            subList.add(this.get(index));
        }
        return subList;
    }  ……  }
```


Capa de negocio
Manejador de lista de valores

```
package com.corej2eepatterns.vlh;
// imports
// sample shows extending ListItertor
public interface ValueListIterator extends ListIterator
{
   // specify convenience methods here
}
```

Capa de negocio
Manejador de lista de valores

```
package com.corej2eepatterns.vlh;
// imports
// typically, this is implemented as an inner class of
// thecustom Value List you implement. In this example,
// ProjectListIterator can be implemented as an inner
// class ofProjectsList. Shown here as a separate class
// for example purposes.
public class ProjectsListIterator implements
ValueListIterator
{
    private List projectsList;
    private int currentIndex = -1;
    private int size = 0;
```


Capa de negocio
Manejador de lista de valores

```
public ProjectsListIterator(List projectsList) {
       this.projectsList = projectsList;
       size = projectsList.size();
       currentIndex=0;
   }

   // implement other methods
   public boolean hasNext() { ... }
   public Object next() { ... }
   public boolean hasPrevious() { ... }
   public Object previous() { ... }
   public int nextIndex() { ... }
```


Capa de negocio
Manejador de lista de valores

```
public int previousIndex() { ... }
   public void remove() { ... }
   public void set(Object o) {...}
   public void add(Object o) { ... }
}
```


Capa de negocio
Manejador de lista de valores

```
package com.corej2eepatterns.dao;
// imports
public class ProjectDAO {
       final private String tableName = "PROJECT";
       // select statement uses fields
       final private String fields = "project_id, name," +
              "project_manager_id, start_date, end_date, " +
              " started, completed, accepted, acceptedDate,"
+             " customer_id, description, status";

       // the methods relevant to the ValueListHandler
       // are shown here.
       // See Data Access Object pattern for other details.
```

Capa de negocio
Manejador de lista de valores

```
      private List findProjects(ProjectTO
projCriteria)
      throws SQLException {

            Statement stmt= null;
            List list = null;
            Connection con = getConnection();
            StringBuffer selectStatement = new
StringBuffer();
            selectStatement.append("SELECT " + fields
+ " FROM " +
                  tableName + "where 1=1");
```


Capa de negocio
Manejador de lista de valores

```
      // append additional conditions to where clause
      // depending on the values specified in
      // projCriteria
            if (projCriteria.projectId != null) {
                  selectStatement.append (" AND
PROJECT_ID = '" +
                        projCriteria.projectId +
"'");
            }
      // check and add other fields to where clause
            ...
```


Capa de negocio
Manejador de lista de valores

```
            try {
                  stmt =
con.prepareStatement(selectStatement);
                  stmt.setString(1, resourceID);
                  ResultSet rs = stmt.executeQuery();
                  list = createResultsList(rs);
                  stmt.close();
            }
            finally {
                  con.close();
            }
            return list;
      }
```


Capa de negocio
Manejador de lista de valores

```
      private List createResultsList(ResultSet rs)
      throws SQLException {
            ArrayList list = new ArrayList();
            while (rs.next()) {
                  int i = 1;
                  ProjectTO proj = new
ProjectTO(rs.getString(i++));
                  proj.projectName = rs.getString(i++);
                  proj.managerId = rs.getString(i++);
                  proj.startDate = rs.getDate(i++);
                  proj.endDate = rs.getDate(i++);
                  proj.started = rs.getBoolean(i++);
                  proj.completed = rs.getBoolean(i++);
```

Capa de negocio
Manejador de lista de valores

```
proj.accepted = rs.getBoolean(i++);
                    proj.acceptedDate = rs.getDate(i++);
                    proj.customerId = rs.getString(i++);
                    proj.projectDescription =
rs.getString(i++);
                    proj.projectStatus = rs.getString(i++);
                    list.add(proj);
            }
            return list;
      }
      ...
}
```


Capa de negocio
Agente de servicio web

- Propósito
  - Se desea proporcionar acceso a uno o más servicios utilizando XML y protocolos web
- También conocido como
  - Web service broker
- Nota
  - El catálogo lo sitúa en integración, pero yo lo veo como un claro patrón de negocio

Capa de negocio
Agente de servicio web

- Motivación
  - Las aplicaciones empresariales exponen sus servicios de grano grueso a través de fachadas de servicio, o de servicios de aplicación
  - Sin embargo, estos servicios pueden no estar preparados para ser expuestos fuera de la aplicación

Capa de negocio
Agente de servicio web

  - Además, es posible que sea necesario componer varios servicios para exponerlos como servicios web:
    - Coordinando interacciones entre uno o más servicios
    - Agregando respuestas
    - Demarcando y/o compensando transacciones

Capa de negocio
Agente de servicio web

- Contexto
  - Se desea reutilizar y exponer distintos servicios a los clientes
  - Se desea monitorizar y potencialmente lmitar el uso de los servicios expuestos, basados en los requisitos de negocio y en el uso de recursos del sistema

Capa de negocio
Agente de servicio web

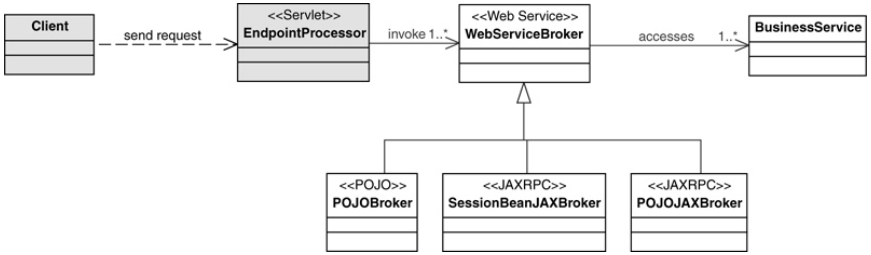
- Los servicios deben ser expuestos utilizados estándares abiertos para permitir la integración de aplicaciones heterogéneas
- Se desea unir la brecha entre requisitos de negocio y capacidades de servicio existentes

Capa de negocio
Agente de servicio web

- Solución
  - Utilizar un agente de servicio web para exponer y negociar uno o más servicios utilizando XML y protocolos web

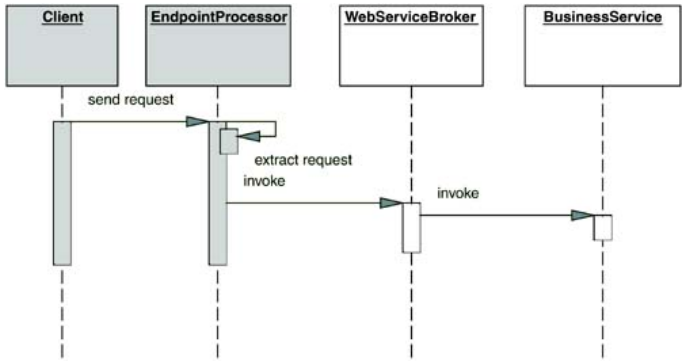
Capa de negocio
Agente de servicio web

- Descripción



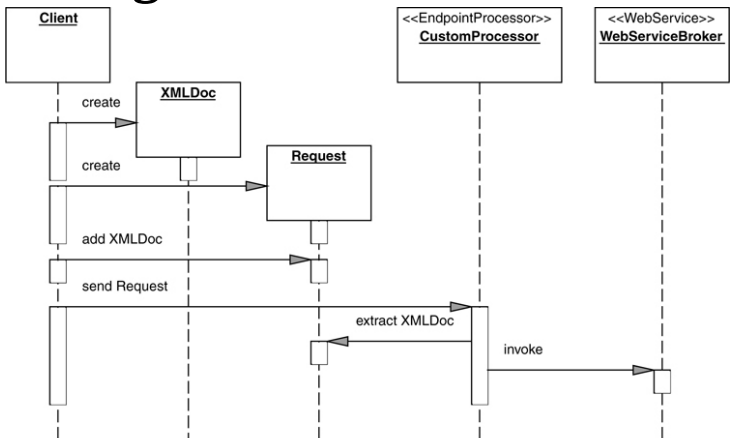
Estructura del patrón agente de servicio web

Capa de negocio
Agente de servicio web



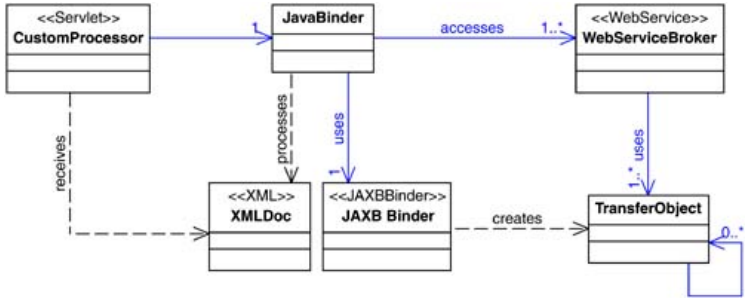
Interacción en el patrón agente de servicio web

Capa de negocio
Agente de servicio web



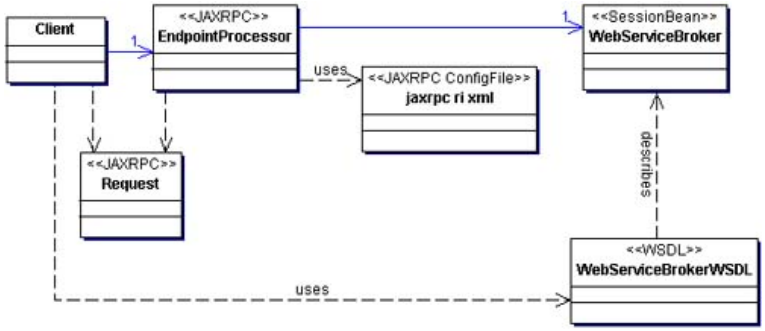
Interacción en el patrón agente de servicio web

Capa de negocio
Agente de servicio web



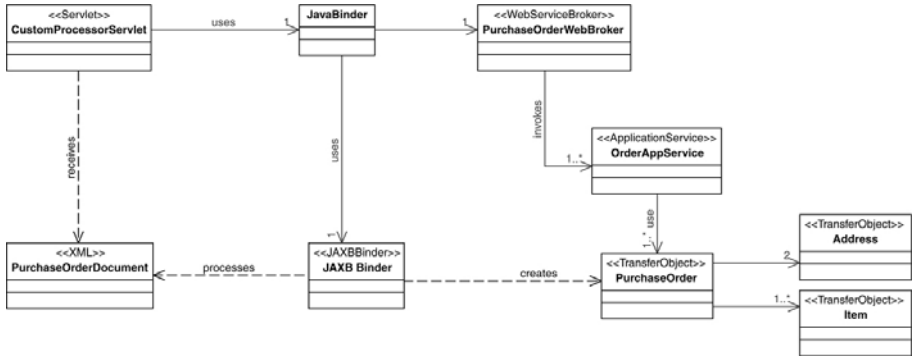
Estructura del patrón para servicios web RESTful

Capa de negocio
Agente de servicio web



Estructura del patrón para servicios web SOAP

Capa de negocio
Agente de servicio web



Ejemplo de agente de servicio web

Capa de negocio
Agente de servicio web

- Consecuencias
  - Ventajas
    - Introduce una capa entre el cliente y el servicio

Capa de negocio
Agente de servicio web

- Ejemplo

```
<?xml version="1.0" encoding="UTF-8"?>
<purchaseOrder orderDate="2003-01-15"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="po2.xsd">
       <shipTo>
              <name>Jo Miller</name>
              <street>Grand View Drive</street>
              <city>Bethesda</city>
              <state>MD</state>
              <zip>20816</zip>
       </shipTo>
```


Capa de negocio
Agente de servicio web

```
<billTo>
       <name>Jane Miller</name>
       <street>Grand View Drive</street>
       <city>Bethesda</city>
       <state>MD</state>
       <zip>20816</zip>
</billTo>
```

Capa de negocio
Agente de servicio web

```
<Items>
      <id>101</id>
      <name>Core J2EE Patterns</name>
      <price>59.99</price>
</Items>
<Items>
      <id>102</id>
      <name>Enterprise Patterns</name>
      <price>59.99</price>
</Items>
```


Capa de negocio
Agente de servicio web

```
<Items>
            <id>103</id>
            <name>WebService Patterns</name>
            <price>59.99</price>
      </Items>
</purchaseOrder>
```


Capa de negocio
Agente de servicio web

```
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CustomProcessorServlet extends
HttpServlet {
      public String getServletInfo() {
            return "Servlet description";
      }
```


Capa de negocio
Agente de servicio web

```
protected void dispatch(HttpServletRequest
      request, HttpServletResponse response,
      String page)
      throws javax.servlet.ServletException,
                  java.io.IOException {
      RequestDispatcher dispatcher =
      getServletContext().getRequestDispatcher(page)
;
            dispatcher.forward(request, response);
      }
```

Capa de negocio
Agente de servicio web

```
public void init() throws ServletException { }

protected void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws
javax.servlet.ServletException,
                  java.io.IOException { }
```


Capa de negocio
Agente de servicio web

```
protected void doPost(HttpServletRequest request,
                  HttpServletResponse response)
                  throws
javax.servlet.ServletException,
                  java.io.IOException {
      String xmlDocument =
      request.getParameter("PurchaseOrder");
      JavaBinder binder = new JavaBinder();
      binder.routeDocument(xmlDocument);
      }
}
```


Capa de negocio
Agente de servicio web

```
import generated.Item;
import generated.PurchaseOrder;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import javax.xml.transform.stream.StreamSource;
import java.io.StringReader;
import java.util.Iterator;
import java.util.List;
```


Capa de negocio
Agente de servicio web

```
public class JavaBinder {
public JavaBinder() { }

private Object parse(String xmlDocument) {
       Object o = null;
       try {
JAXBContext jc = JAXBContext.newInstance("generated");
             Unmarshaller u = jc.createUnmarshaller();
             o = u.unmarshal(new StreamSource(
             new StringReader(xmlDocument)));
             } catch (JAXBException e) {
             }
             return o;     }
```

Capa de negocio
Agente de servicio web

```
public void routeDocument(String xmlDocument) {
       Object o = parse(xmlDocument);
       if (o instanceof PurchaseOrder) {
              PurchaseOrder po = (PurchaseOrder) o;
              printIt(po);
              PurchaseOrderWebBroker broker =
              new PurchaseOrderWebBroker();
              broker.placeOrder(po);
              }
       }
```


Capa de negocio
Agente de servicio web

```
private void printIt(PurchaseOrder po) {
       System.out.println(po.getBillTo());
       System.out.println(po.getShipTo());
       System.out.println(po.getOrderDate());
       List items = po.getItems();
       Iterator i = items.iterator();
       while (i.hasNext()) {
              Item item = (Item) i.next();
              System.out.println("Id : " + item.getId()
              + " Name : " + item.getName() +
              " Price : " + item.getPrice());
              }
       }  }
```


Capa de negocio
Agente de servicio web

```
import generated.PurchaseOrder;

public class PurchaseOrderWebBroker {
 public void placeOrder(PurchaseOrder po) {
       // Do security checks here, if necessary
       OrderAppService as = new OrderAppService();
       as.placeOrder(po);
       }
}
```


Capa de negocio
Agente de servicio web

```
import generated.Address;
import generated.PurchaseOrder;
import java.util.List;
public class OrderAppService {
public OrderAppService() { }
 public void placeOrder(PurchaseOrder purchaseOrder) {
              Address shipTo = purchaseOrder.getShipTo();
              Address billTo = purchaseOrder.getBillTo();
              List items = purchaseOrder.getItems();

              // Update data store
       }
}
```

Capa de integración
DAO

- Propósito
  - Se desea encapsular la manipulación y acceso a datos en una capa separada
- También conocido como
  - *Data access object*
  - Objeto de acceso a datos

Capa de integración
DAO

- Motivación
  - Si se tiene un mecanismo de persistencia avanzado, el acceso a datos queda totalmente oculto tras un API que proporciona objetos del negocio
  - Si no se dispone los servicios de aplicación, fachadas y otros ayudantes tendrán que acceder a datos de negocio en el almacén persistente

Capa de integración
DAO

  - Almacenes persistentes pueden ser SGBDR, ficheros, repositorios XML, sistemas heredados, etc.
  - Manejar estos datos en estos almacenes fuerza a:
    - Conocer los mecanismos de acceso del sistema de gestión de datos (p.e., base de datos, sistema operativo, etc.)
    - Conocer la representación de los datos en el sistema de gestión de datos (p.e., columnas, elementos, bytes, etc.)

Capa de integración
DAO

  - Introducir lógica de persistencia en la lógica de negocio genera un fortísimo acoplamiento entre negocio y los mecanismos de persistencia

Capa de integración
DAO

- Contexto
  - Se desea implementar mecanismos de acceso y manipulación de datos en un almacén persistente
  - Se desea desacoplar la implementación del almacén persistente del resto de la aplicación
  - Se desea proponer un API uniforme de acceso a datos para varios tipos de fuentes de datos, tales como SGBDR, ficheros, repositorios XML, etc.

Capa de integración
DAO

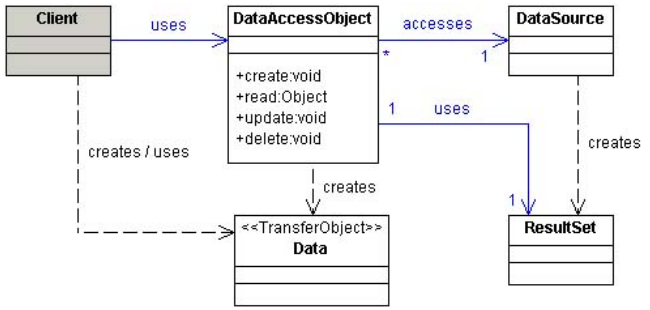
  - Se desea organizar la lógica de acceso a datos y encapsular características propietarias para facilitar la mantenibilidad y portabilidad

Capa de integración
DAO

- Solución
  - Utilizar un DAO para abstraer y encapsular todo el acceso al almacén persistente
  - El DAO gestiona la conexión con la fuente de datos para obtener y almacenar datos

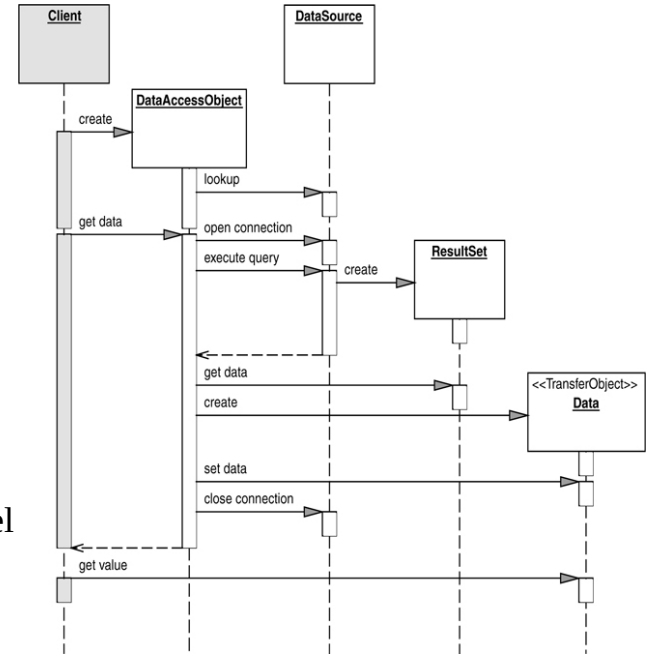
Capa de integración
DAO

- Descripción

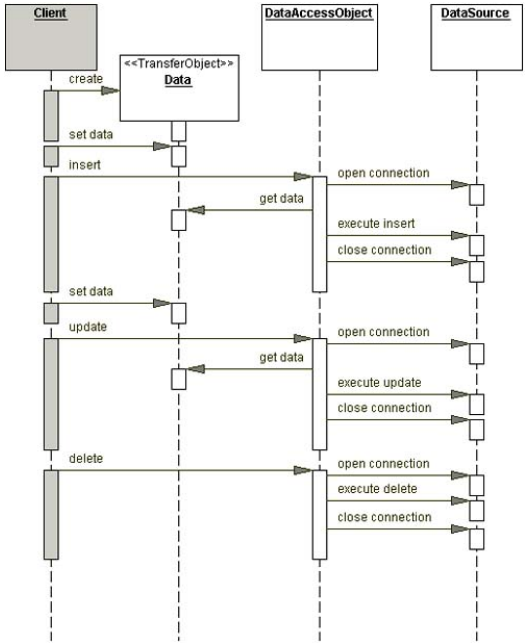


Estructura del patrón DAO

Interacción en el patrón DAO



Interacción en el patrón DAO



Capa de integración DAO

- Consecuencias
  - Ventajas:
    - Permite transparencia
    - Proporciona una visión orientada a objetos y encapsula esquemas de bases de datos
    - Permite una migración más sencilla

Capa de integración DAO

- Código de ejemplo

```
public interface DAOUsuario {
  //podría haberse considerado un DAO para cada
  //operación
  public Integer insertaUsuario(TUsuario tUsuario);
  public Boolean daDeBajaUsuario(Integer id);
  public TUsuario obtenUsuario(Integer id);
  public Boolean modificaUsuario(TUsuario tUsuario);

}
```

Capa de integración
DAO

```
public class DAOUsuarioImp implements DAOUsuario {
……
   public Boolean daDeBajaUsuario(Integer idInteger)
   {
       boolean resultado= true;
       int id= idInteger.intValue();

       //conexión con la base de datos
```


Capa de integración
DAO

```
       Sring plantilla= "UPDATE usuario SET
                        activo=false WHERE id=?";
       PreparedStatement pstmt=
                 con.prepareStatement(plantilla);

       pstmt.setInt(1, id);
       resultado= (pstmt.executeUpdate() > 0);

       //cerrar conexión y tratar excepciones

       return new Boolean(resultado);
   }
……..
}
```


Capa de integración
DAO

- Aunque en estas transparencias se obvia, es fundamental que los DAOs capturen y lancen las excepciones correspondientes al acceder a los recursos externos
- Así, la capa de negocio sabrá qué ha sucedido si ha habido algún tipo de fallo en dicho acceso

Capa de integración
Activador de servicio

- Propósito
  - Se desea invocar servicios de manera asíncrona
- También conocido como
  - Service activator

Capa de integración
Activador de servicio

- Motivación
  - En aplicaciones empresariales la mayor parte del procesamiento se lleva a cabo de manera síncrona
  - Así, un cliente invoca un servicio de negocio y espera hasta que el servicio termine el procesamiento
  - Sin embargo, a veces, el procesamiento del servicio puede utilizar mucho tiempo y recursos

Capa de integración
Activador de servicio

  - El servicio incluso podría extenderse por otras aplicaciones externas
  - No es razonable que las aplicaciones clientes esperen a que estos servicios de larga duración terminen de ejecutarse

Capa de integración
Activador de servicio

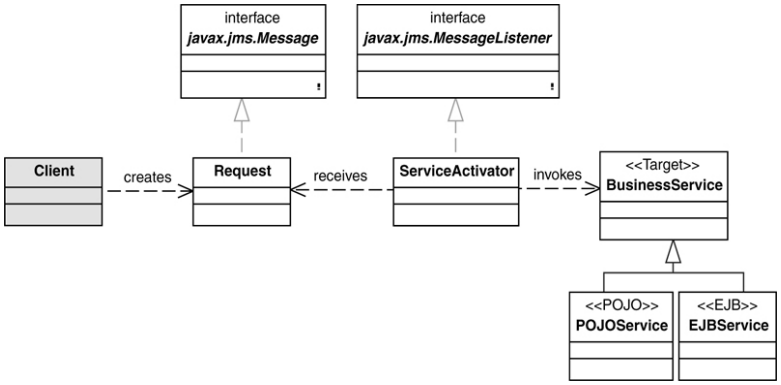
- Contexto
  - Se desea invocar servicios de negocio de una manera asíncrona
  - Se desea integrar mensajería publicar/suscribir y punto a punto para posibilitar servicios de procesamiento asíncronos
  - Se desea llevar a cabo una tarea de negocio formado por distintas tareas de negocio

Capa de integración
Activador de servicio

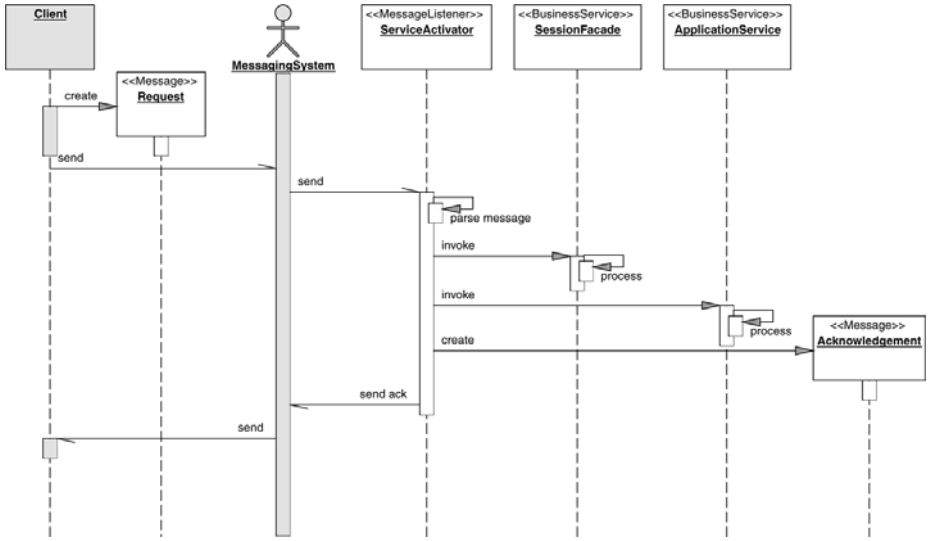
- Solución
  - Utilizar un activador de servicio para recibir peticiones asíncronas e invocar uno o más servicios de negocio

Capa de integración
Activador de servicio

- Descripción



Estructura del patrón activador de servicio



Interacción en patrón activador de servicio

Capa de integración
Activador de servicio

- Consecuencias
  - Ventajas
    - Integra JMS (Java Message Service) en aplicaciones empresariales
    - Proporciona procesamiento asíncrono para cualquier componente de la capa de negocio
    - Permite tener listeners JMS independientes de servidores de aplicaciones

Capa de integración
Activador de servicio

- Ejemplo

```
// imports. . .
public class OrderServiceActivator implements
javax.jms.MessageListener{
// Queue session and receiver: see JMS API for details
       private QueueSession orderQueueSession;
       private QueueReceiver orderQueueReceiver;
// Note: values should come from property files or
// environment. Hard coded here for illustration only
       private String connectionFactoryName =
                     "PendingOrdersQueueFactory";
       private String queueName = "PendingOrders";
```

Capa de integración
Activador de servicio

```
// use a service locator to locate administered JMS
// components such as a Queue or a Queue Connection
// factory
      private ServiceLocator serviceLocator;
      private QueueConnectionFactory
queueConnectionFactory;
```


Capa de integración
Activador de servicio

```
private QueueConnection queueConnection;

public OrderServiceActivator(
                  String connectionFactoryName,
String queueName) {
            super();
            this.connectionFactoryName =
connectionFactoryName;
            this.queueName = queueName;
            startListener();
      }
```


Capa de integración
Activador de servicio

```
private void startListener() {
            try {
                  serviceLocator =
ServiceLocator.getInstance();
                  queueConnectionFactory =

      serviceLocator.getQueueConnectionFactory(
                              connectionFactoryName);
                  queueConnection =

      queueConnectionFactory.createQueueConnection();
```


Capa de integración
Activador de servicio

```
// See JMS API for method usage and arguments
                  orderQueueSession =
            queueConnection.createQueueSession (...);
                  Queue ordersQueue =
                  serviceLocator.getQueue(queueName);
                  orderQueueReceiver =
      orderQueueSession.createReceiver(ordersQueue);
      orderQueueReceiver.setMessageListener(this);
            }
            catch (JMSException excp) {
                  // handle error
            }
      }
```

Capa de integración
Activador de servicio

```
// The JMS API specifies the onMessage method in the
// javax.jms.MessageListener interface.
// This method is asynchronously invoked
// when a message arrives on the Queue being
// listened to by the ServiceActivator.
// See JMS Specification and API for more details.
public void onMessage(Message message) {
            try {
      // parse Message msg. See JMS API for Message.
      // Get the order object from the message
                  ObjectMessage objectMessage =
            (javax.jms.ObjectMessage) message;
      Order order = (Order) objectMessage.getObject();
```


Capa de integración
Activador de servicio

```
// Use the Business Delegate for Order Processor
// Session Façade to invoke order processing
            OrderProcessorDelegate orderProcessorBD =
            new OrderProcessorDelegate();
            orderProcessorBD.processOrder(order);
                  // send any response here...
            }
            catch (JMSException jmsexcp) {
                  // Handle JMSExceptions, if any
                  // Send error response, throw
runtime exception
            }
```


Capa de integración
Activador de servicio

```
            catch (Exception excp) {
      // Handle any other exceptions
      // Send error response, throw runtime exception
            }     }
  public void close() {
            try {  // cleanup before closing
      orderQueueReceiver.setMessageListener (null);
            orderQueueSession.close();
            }
            catch(Exception excp) {
            // Handle exception - Failure to close
            }
      } }
```


Capa de integración
Activador de servicio

```
// imports...
public class OrderSenderAppService {
      // Queue session and sender: see JMS API for
details
      private QueueSession orderQueueSession;
      private QueueSender orderQueueSender;
// These values could come from some property files
      private String connectionFactoryName =
                  "PendingOrdersQueueFactory";
      private String queueName = "PendingOrders";
```

Capa de integración
Activador de servicio

```
      // use a service locator to locate administered
      // JMS components such as a Queue or a Queue.
      // Connection factory
      private JMSServiceLocator serviceLocator;
      . . .
      // method to initialize and create queue sender
private void createSender() {
            try {
            // using ServiceLocator and getting Queue
            // Connection Factory is similar to the
            // Service Activator code.
                  serviceLocator =
ServiceLocator.getInstance();
```


Capa de integración
Activador de servicio

```
      queueConnectionFactory =
      serviceLocator.getQueueConnectionFactory(
                        connectionFactoryName);
      queueConnection =
      queueConnectionFactory.createQueueConnection();
      // See JMS API for method usage and arguments
      orderQueueSession =
            queueConnection.createQueueSession(. . .);
      Queue ordersQueue =
            serviceLocator.getQueue(queueName);
      orderQueueSender =
            orderQueueSession.createSender(ordersQueue);
            }
```


Capa de integración
Activador de servicio

```
catch(Exception excp) {
                  // Handle exception - Failure to
create sender
            }
      }
// method to dispatch order to fulfillment service
    // for asynchronous processing

public void sendOrder(Order order) {
            try {
      // create a new Message to send Order object
                  ObjectMessage orderObjectMessage =
```


Capa de integración
Activador de servicio

```
      queueSession.createObjectMessage(order);
      // set object message properties and delivery
      // mode as required.
      // See JMS API for ObjectMessage
      // Set the Order into the object message
      orderObjectMessage.setObject(order);
      // send the message to the Queue
      orderQueueSender.send(orderObjectMessage);
                  . . .
      } catch (Exception e) {
                  // Handle exceptions      }
            . . .
      }. . .
```

Capa de integración
Activador de servicio

```
public void close() {
              try {
                     // cleanup before closing

       orderQueueReceiver.setMessageListener(null);
                     orderQueueSession.close();
              }
              catch(Exception excp) {
                     // Handle exception - Failure to
close
              }
       }
}
```


Concurrencia en persistencia
Introducción

- La concurrencia es uno de los aspectos más complejos en desarrollo de software
- Aparece cuando se tienen distintos procesos o hebras manipulando los mismos datos
- En aplicaciones empresariales lo resolvemos con los transaction managers

Concurrencia en persistencia
Problemas

- *Pérdida de actualizaciones:*
  - La hebra A toma el control y lee el empleado 44
  - La hebra B toma el control, despide al empleado 44 y termina
  - La hebra A sube el sueldo al empleado 44 y termina
  - El empleado 44 no es despedido y le suben el sueldo

Concurrencia en persistencia
Problemas

- *Lectura inconsistente*
  - La hebra A toma el control y lee la tabla proyectos
  - La hebra B toma el control da de baja al proyecto 27, despide a sus empleados y termina
  - La hebra A toma el control y lee el fichero de empleados
  - La hebra A va a tener un proyecto 27 con todos sus empleados despedidos

Concurrencia en persistencia Problemas

- Ambos problemas son un ejemplo de fallo en la *corrección*, y se debe al acceso concurrente a los mismos datos
- Pueden resolverse eliminando la concurrencia, pero eso nos llevaría a un problema de *viveza*: la cantidad de actividad concurrente que puede llevarse a cabo simultánemamente

Concurrencia en persistencia Problemas

- Corrección y viveza son atributos en tensión que tendrán que equilibrarse en cada aplicación

Concurrencia en persistencia Contextos de ejecución

- En una aplicación empresarial hay dos contextos de ejecución importantes: la petición (request) y la sesión
- Una *petición* es una única llamada del mundo exterior a nuestra aplicación que puede ser contestada con una respuesta (response)

Concurrencia en persistencia Contextos de ejecución

- Una *sesión* es una interacción de larga duración entre un cliente y un servidor. Normalmente incluirá varios ciclos petición/respuesta
- Normalmente la sesión tiene lugar dentro de la misma *hebra* de ejecución, que es un mecanismo de ejecución concurrente más ligero que el *proceso*

Concurrencia en persistencia Contextos de ejecución

- En lo referente a datos, hay un contexto importante: la *transacción*, que agrupa acciones contra la base de datos:
  - *Transacción del sistema*: acciones de la aplicación contra la base de datos
  - *Transacción de negocio*: acciones del usuario contra la aplicación

Concurrencia en persistencia Aislamiento e inmutabilidad

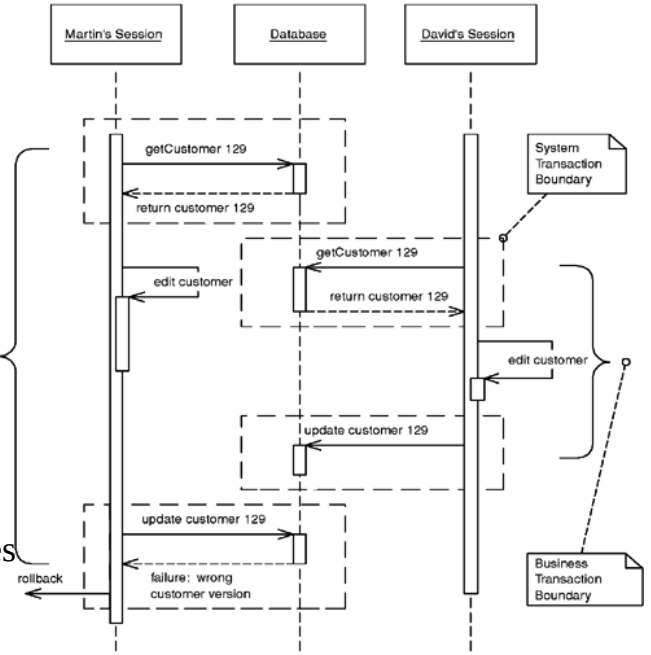
- El *aislamiento* divide los datos de tal forma que cualquier elemento suyo sólo pueda ser accedido por un agente activo (i.e. proceso o hebra)
- La *inmutabilidad* fuerza a que un dato no pueda ser cambiado, evitando así problemas de concurrencia

Concurrencia en persistencia Aislamiento e inmutabilidad

- Una forma sencilla de inmutabilidad es permitir accesos de sólo lectura: los datos no son inmutables, pero los clientes sólo pueden leerlos

Concurrencia en persistencia Control optimista y pesimista

- ¿Qué hacer cuando tenemos datos mutables que no pueden ser aislados?

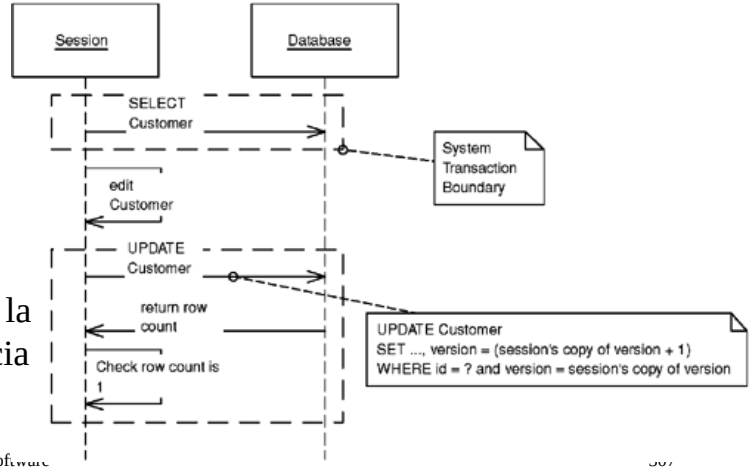


Problemas de acceso y modificaciones concurrentes

Concurrencia en persistencia Control optimista y pesimista

- Podemos hacer un control de la concurrencia optimista y pesimista
- El control de la concurrencia *optimista* permite cambios sin control, pero garantiza que el cambio se realiza sobre el objeto original (p.e. con un código de versión)

Concurrencia en persistencia Control optimista y pesimista



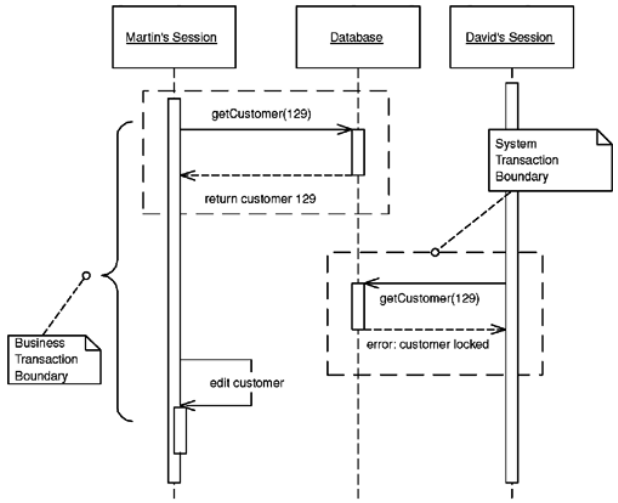
Control de la concurrencia pesimista

Concurrencia en persistencia Control optimista y pesimista

- El control de la concurrencia *pesimista* bloquea los datos, evitando modificaciones ajenas al agente que los bloqueo

Concurrencia en persistencia
Control optimista y pesimista

Control pesimista de la concurrencia



Concurrencia en persistencia
Control optimista y pesimista

- Podríamos decir que un control optimista detecta conflictos, y un control pesimista los evita

Concurrencia en persistencia
Control optimista y pesimista

- El control pesimista reduce la concurrencia y puede producir bloqueos
  - La hebra A lee la tabla 1 y la bloquea
  - La hebra B lee la tabla 2 y la bloquea
  - La hebra A intenta leer la tabla 2 y no puede
  - La hebra B intenta leer la tabla 1 y no puede

Concurrencia en persistencia
Control optimista y pesimista

- El control optimista puede producir conflictos problemáticos:
  - La hebra A hace una venta de los productos 1, 2 y 3
  - Según va leyendo los productos, otras tres hebras (X, Y, Z) que hacen venta también leen simultáneamente esos productos (1, 2 y 3)
  - La hebra A modifica los tres productos
  - Las otras tres hebras encuentran problemas y tienen que abortar toda su venta en curso

Concurrencia en persistencia Control optimista y pesimista

- Si los conflictos son poco probables o de poca consecuencia, es mejor un control optimista
- En otro caso, es mejor el pesimista

Concurrencia en persistencia Control optimista y pesimista

- Otro problema es el de las *lecturas inconsistentes*
- El control pesimista puede extenderse a la lectura, para evitarla
- El control optimista fuerza a que exista un marcador de versión para datos compartidos, y a contrastarlo

Concurrencia en persistencia Control optimista y pesimista

- Otra forma de solucionarlo es que el almacen de datos permita *lecturas temporales*, donde los datos van *marcados* con algún tipo de marca temporal o etiqueta inmutable. No es muy común

Concurrencia en persistencia Transacciones

- La principal herramienta para controlar la concurrencia en aplicaciones empresariales son las transacciones
- Una transacción es una agrupación de acciones que debe tener las propiedades *ACID:*

Concurrencia en persistencia Transacciones

- *Atomicity*: cada paso en la secuencia de acciones dentro de una transacción debe completarse con éxito o debe echarse para atrás
- *Consistency*: los recursos de un sistema deben estar en un estado consistente, no corrupto al principio y al final de la transacción
- *Isolation*: el resultado de una transacción no debe ser visible a ninguna otra transacción abierta hasta que la transacción termine con éxito

Concurrencia en persistencia Transacciones

- *Durability*: cualquier resultado de una transacción comprometida debe ser permanente, es decir, debe sobrevivir a un accidente de cualquier tipo

- En general, puede haber más recursos que bases de datos participando en una transacción (colas de mensajes, periféricos, etc.)

Concurrencia en persistencia Transacciones

- Así un *recurso transaccional* es cualquier cosa que utiliza transacciones para controlar la concurrencia (e.g. un SGBDR)
- Hay varios tipos de transacciones:
  - *Transacción larga*: una transacción que se expande a través de múltiples peticiones
  - *Transacción petición*: una transacción que sólo dura durante el procesamiento de la petición

Concurrencia en persistencia Transacciones

  - *Transacción de última hora*: es una transacción de petición que hace todas las lecturas posibles fuera de ella y que la abre para las actualizaciones

- Cuando se hace una transacción se hacen bloqueos y se produce una *escalada de bloqueos*, donde hay múltiples bloqueos de recursos, dificultando el procesamiento

Concurrencia en persistencia Transacciones

- También puede interesar reducir el aislamiento de la transacción para mejorar la viveza
- Si hay pleno aislamiento las transacciones son *serializables*: las transacciones se ejecutan como si no hubiera otras transacciones en marcha

Concurrencia en persistencia Transacciones

- Los problemas asociados al aislamiento son:
  - *Lecturas sucias*: las lecturas de una transacción se ven afectadas por modificaciones no comprometidas de otra
    - T1 lee la tabla Proyecto: salen cuatro
    - T2 inserta en la tabla Proyecto
    - T1 lee la tabla Proyecto: salen cinco
    - T2 hace un rollback

Concurrencia en persistencia Transacciones

  - *Lecturas irrepetibles*: las modificaciones o borrados comprometidos en una transacción son visibles en otras:
    - T1 lee la tabla Proyecto: salen cuatro
    - T2 borra en la tabla Proyecto y hace commit
    - T1 lee en la tabla Proyecto: salen tres

Concurrencia en persistencia Transacciones

  - *Lecturas fantasma*: las inserciones comprometidas en una transacción son visibles a otras en las que el predicado de selección se ve afectado:
    - T1 lee la tabla Proyectos dirigidos por el departamento de I+D: salen dos
    - T2 inserta un Proyecto dirigido por el departamento de I+D
    - T1 lee la tabla Proyectos: salen tres

Concurrencia en persistencia Transacciones

– En base a esto, los niveles de aislamiento permiten:

	lecturas sucias	**lecturas no repetibles**	**lecturas fantasma**
read uncommitted	posible	posible	posible
read commited	imposible	posible	posible
repeatable read	imposible	imposible	posible
serializable	imposible	imposible	imposible

Niveles de aislamiento

Concurrencia en persistencia Transacciones

- Hay otro tipo de transacciones:
  – *De sistema*: se llevan a cabo teniendo en mente un sistema concreto (e.g. una venta donde la validación de cliente y stock se hace al cerrarla)
  – *De negocio*: se llevan a cabo teniendo en cuenta una regla de negocio (e.g. una venta donde la validación de cliente y stock se hace al añadir al carrito)

Concurrencia en persistencia Transacciones

- Al final, las transacciones de negocio suelen ser largas (más de una petición) y las del sistema cortas (una única petición)
- Se suele simular una de negocio con varias de sistemas, lo que puede generar el problema de la *concurrencia offline* ya que las del negocio también deberían ser ACID:

Concurrencia en persistencia Transacciones

– La atomicidad y durabilidad son sencillas (el commit/rollback se hace en cerrar venta)
– Lo difícil de garantizar es:
  - El aislamiento (no puedo garantizar que el cliente siga activo a la hora de cerrar la venta)
  - Esto genera fallos de inconsistencia (puedo hacer una venta a un cliente dado de baja)

Concurrencia en persistencia Transacciones

- Otro problema es la *concurrencia en el servidores web y de aplicaciones*
- No afecta directamente al almacén persistente, pero sí a los datos asociados a las sesiones y/o peticiones

Concurrencia en persistencia Transacciones

- Hay tres opciones:
  - *Proceso por sesión*: cada sesión es un proceso. Hay aislamiento entre sesiones, pero es muy costoso
  - *Proceso por petición*: cada petición es un proceso. Hay aislamiento entre peticiones, pero sigue siendo costoso
  - *Hebra por petición*: cada petición está gestionada por una hebra. No hay aislamiento entre peticiones, pero es menos costoso

Concurrencia en persistencia Transacciones

- Lo normal es que sea hebra por petición
- Esto fuerza a que los singleton estén preparados preparados para trabajar en entornos concurrentes

Capa de integración Almacén del dominio

- Propósito
  - Se desea separar la persistencia del modelo de objetos
- También conocido como:
  - Domain store
  - Unit of work + Query object + Data mapper + Table data gateway + Dependent mapping + Domain model + Data transfer object + Identity map + Lazy load

Capa de integración
Almacén del dominio

- Motivación
  - Muchos sistemas tienen un modelo de objetos complejo que requiere sofisticadas estrategias de persistencia
  - Estas estrategias deberían ser independientes de los objetos del negocio, para no acoplarlos con un almacén concreto

Capa de integración
Almacén del dominio

  - Así, deben resolverse cuatro problemas simultáneos:
    - Persistencia
    - Carga dinámica
    - Gestión de transacciones
    - Concurrencia

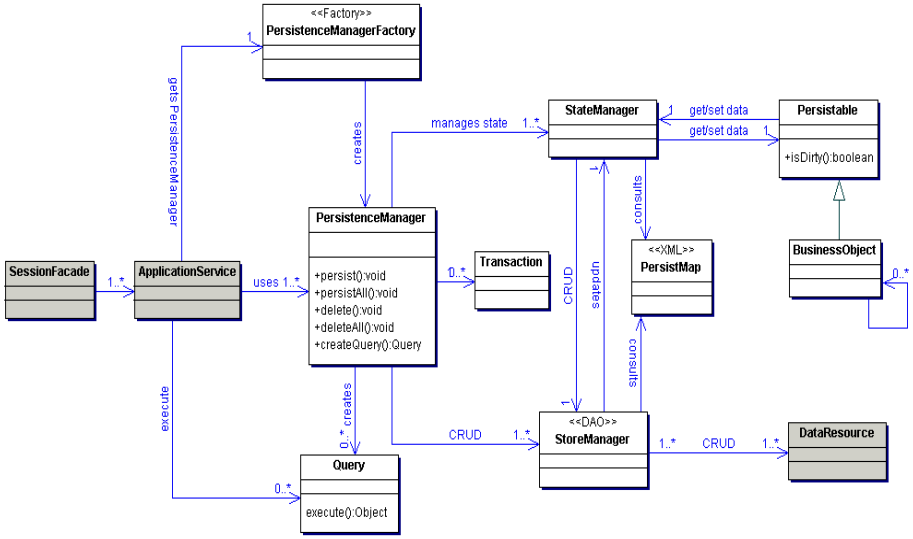
Capa de integración
Almacén del dominio

- Contexto
  - Se desea omitir detalles de persistencia en los objetos del negocio
  - La aplicación podría ejecutarse en un contenedor web
  - El modelo de objetos utiliza herencia y relaciones compleja

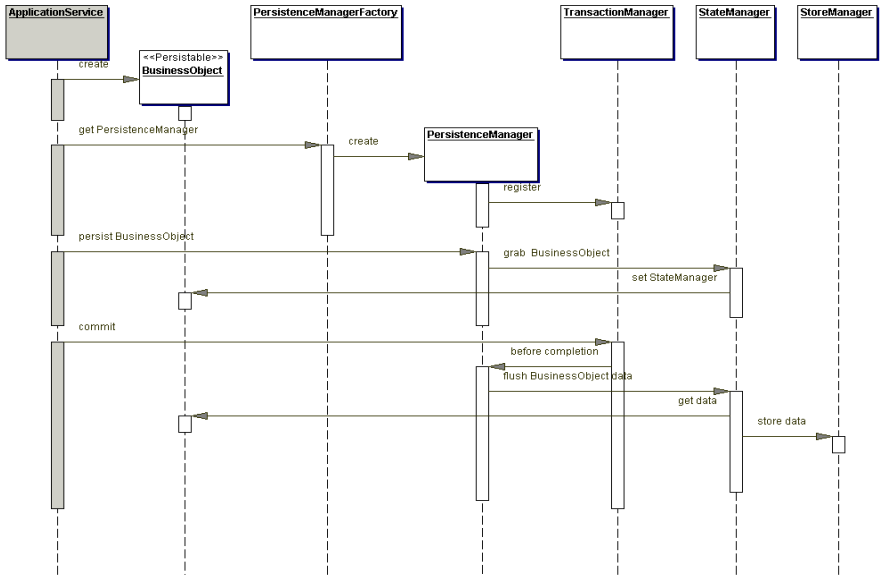
Capa de integración
Almacén del dominio

- Solución
  - Utilizar un almacén del dominio para persistir de manera transparente un modelo de objetos

- Descripción

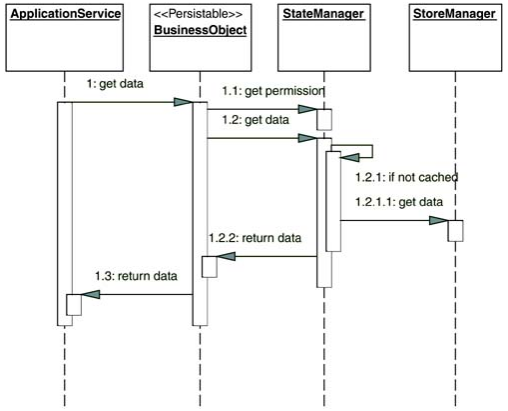


Estructura del patrón almacén del dominio

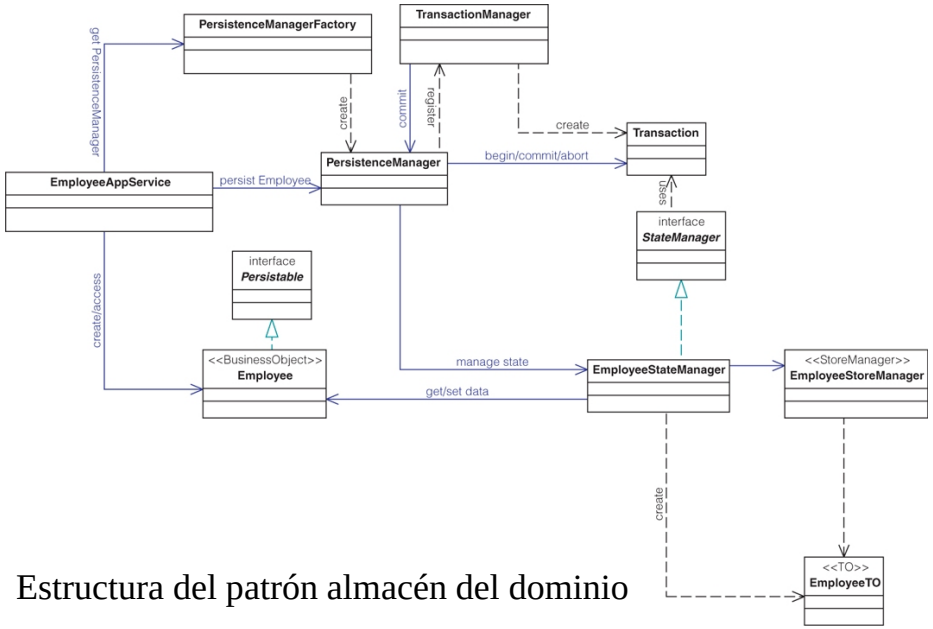


Persistencia de un objeto del negocio

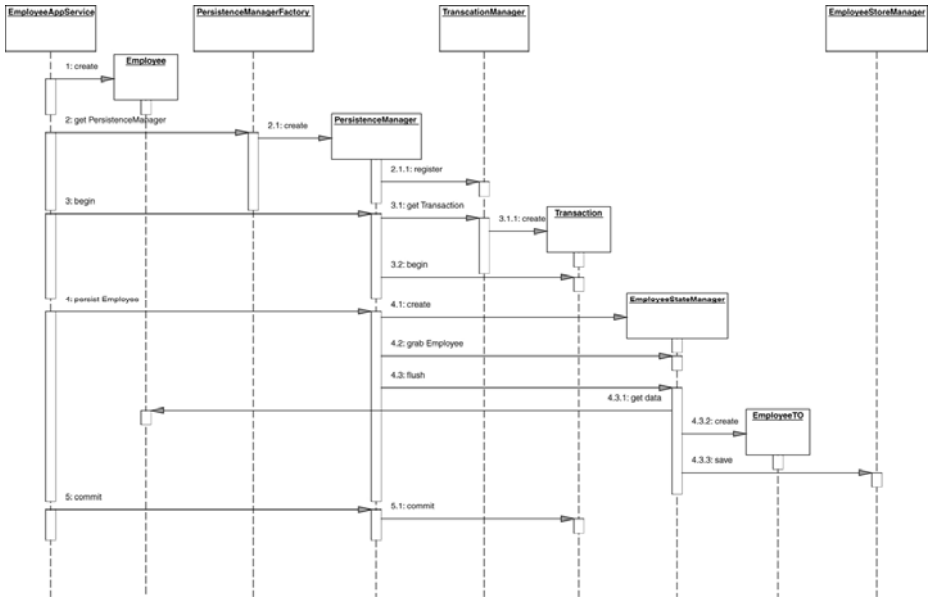
Capa de integración
Almacén del dominio



Acceso a atributos simples de un objeto del negocio



Estructura del patrón almacén del dominio



Interacción en el patrón almacén del dominio

Capa de integración
Almacén del dominio

- Consecuencias
  - Ventajas:
    - Resuelve:
      - Persistencia
      - Carga dinámica
      - Transacciones
      - Concurrencia
    - Mejora el entendimiento de los marcos de persistencia
    - Mejora la prueba del modelo de objetos persistente
    - Separa el modelo de objetos de negocio de lógica de persistencia

Capa de integración
Almacén del dominio

  - Inconvenientes:
    - Crear un marco de persistencia a medida es una tarea compleja
    - La carga y almacenamiento de un árbol de objetos requiere técnicas de optimización
    - Un marco de persistencia en toda regla podría ser excesivo para un modelo de objetos pequeño

Capa de integración
Almacén del dominio

- Ejemplo

```
import javax.transaction.*;
public class EmployeeApplicationService {
 public String createEmployee(String lastName,
       String firstName, String ss, float salary,
           String jobClassification, String geography) {
       String id = null;
       String divisionId = null;
       // Create new id
       divisionId =
             getDivisionId(jobClassification, geography);
```

Capa de integración
Almacén del dominio

```
// Create Employe
            Employee e = new Employee(id, lastName,
firstName,
                                ss, salary, divisionId);
      PersistenceManagerFactory factory =
            PersistenceManagerFactory.getInstance();
      PersistenceManager manager =
                        factory.getPersistenceManager();
            try {
                  manager.begin();
                  e = (Employee) manager.persistNew(e);
                  manager.commit();
            } catch (SystemException e1) {
```


Capa de integración
Almacén del dominio

```
            } catch (NotSupportedException e1) {
            } catch (HeuristicRollbackException e1) {
            } catch (RollbackException e1) {
            } catch (HeuristicMixedException e1) {
            }
            return id;
      }
```


Capa de integración
Almacén del dominio

```
public void setEmployeeSalary(String id, float salary)
{
      PersistenceManagerFactory factory =
            PersistenceManagerFactory.getInstance();
      PersistenceManager manager =
                  factory.getPersistenceManager();
      Employee e = manager.getEmployee(id);

      if (e != null) {
                  e.setSalary(salary);
            }
```


Capa de integración
Almacén del dominio

```
      try {
                  manager.begin();
                  e = (Employee) manager.persist(e);
                  manager.commit();
            } catch (SystemException e1) {
            } catch (NotSupportedException e1) {
            } catch (HeuristicRollbackException e1) {
            } catch (RollbackException e1) {
            } catch (HeuristicMixedException e1) {
            }
      }
}
```

Capa de integración
Almacén del dominio

```
import javax.transaction.*;
public class EmployeeApplicationService {
    public String createEmployee(String lastName,
                                 String firstName,
                                 String ss,
                                 float salary,
                                 String jobClassification,
                                 String geography) {
        String id = null;
        String divisionId = null;
        // Biz Logic
```


Capa de integración
Almacén del dominio

```
 // Create new id
       id = getNewId();
       divisionId = getDivisionId( jobClassification );
// Create Employe
        Employee e = new Employee(id, lastName,
                    firstName, ss, salary, divisionId);
        PersistenceManagerFactory factory =
        PersistenceManagerFactory.getInstance();
        PersistenceManager manager =
factory.lookupPersistenceManager();
```


Capa de integración
Almacén del dominio

```
try {
            manager.begin();
            manager.persist(e);
            manager.commit();
        } catch (SystemException e1) {
        } catch (NotSupportedException e1) {
        } catch (HeuristicRollbackException e1) {
        } catch (RollbackException e1) {
        } catch (HeuristicMixedException e1) {
        }
        return id;
    }
}
```


Capa de integración
Almacén del dominio

```
public interface Persistable { }

public class Employee implements Persistable {
       protected String id;
       protected String firstName;
       protected String lastName;
       protected String ss;
       protected float salary;
       protected String divisionId;

       public Employee(String id) {
              this.id = id;
       }
```

Capa de integración
Almacén del dominio

```
public Employee(String id, String lastName,
       String firstName, String ss, float salary,
                          String divisionId) {
              this.firstName = firstName;
              this.lastName = lastName;
              this.firstName = firstName;
              this.ss = ss;
              this.salary = salary;
              this.divisionId = divisionId;
       }
```


Capa de integración
Almacén del dominio

```
       public void setId(String id) {
              this.id = id; }
       public void setFirstName(String firstName) {
              this.firstName = firstName; }
       public void setLastName(String lastName) {
              this.lastName = lastName; }
       public void setSalary(float salary) {
              this.salary = salary;      }
       public void setDivisionId(String divisionId) {
              this.divisionId = divisionId; }
       public void setSS(String ss) {
              this.ss = ss; }
. . . }
```


Capa de integración
Almacén del dominio

```
public class EmployeeStateManager implements
StateManager {
       private final int ROW_LEVEL_CACHING = 1;
       private final int FIELD_LEVEL_CACHING = 2;
       int cachingType = ROW_LEVEL_CACHING;
       boolean isNew;
       private Employee employee;
       private PersistenceManager pm;
```


Capa de integración
Almacén del dominio

```
public EmployeeStateManager(PersistenceManager pm,
                     Employee employee, boolean isNew )
{
              this.pm = pm;
              this.employee = employee;
              this.isNew = isNew;
       }
```

Capa de integración
Almacén del dominio

```
public void flush() {
       if (pm.isDirty(employee)) {
              EmployeeTO to = new EmployeeTO(employee.id,
                     employee.lastName, employee.firstName,
                     employee.ss, employee.salary,
                     employee.divisionId);
              EmployeeStoreManager storeManager =
                           new EmployeeStoreManager();
              if (isNew) {  storeManager.storeNew(to);
                           isNew = false;
                     } else { storeManager.update(to); }
                     pm.resetDirty(employee);
              }
       }
```


Capa de integración
Almacén del dominio

```
public void load() {
              EmployeeStoreManager storeManager =
                           new EmployeeStoreManager();
              EmployeeTO to = storeManager.load(employee.id);
              updateEmployee(to);
       }
```


Capa de integración
Almacén del dominio

```
public void load(int field) {
       if (fieldNeedsReloading(field)) {
                     EmployeeStoreManager storeManager =
                     new EmployeeStoreManager();
                     if (cachingType ==
                     FIELD_LEVEL_CACHING) {
Object o = storeManager.loadField(employee.id, field);
                      updateEmployee( field, o );
                     } else {
                           EmployeeTO to =
storeManager.load(employee.id);
                           updateEmployee(to);
                     }
              } }
```


Capa de integración
Almacén del dominio

```
private boolean fieldNeedsReloading(int field) {
// Caching and valid data rule apply here
// data can be cached at the field or the row level
       switch (field) {
                     case EmployeeStateDelegate.LAST_NAME:
              if (employee.lastName == null) return true;
                           break;
                     case EmployeeStateDelegate.FIRST_NAME:
              if (employee.firstName == null) return true;
                           break;
```

Capa de integración
Almacén del dominio

```
            case EmployeeStateDelegate.DIVISION_ID:
                  String did = employee.divisionId;
      if (did == null || did.indexOf("99-") == -1)
                        return true;
                  break;
            case EmployeeStateDelegate.SS:
                      if (employee.ss == null) return true;
                        break;
            case EmployeeStateDelegate.SALARY:
                  if (employee.salary == 0.0) return true;
                        break;
                  }
            return false;       }
```


Capa de integración
Almacén del dominio

```
private void updateEmployee(EmployeeTO to) {
              employee.id = to.id;
              employee.lastName = to.lastName;
              employee.firstName = to.firstName;
              employee.ss = to.ss;
              employee.salary = to.salary;
              employee.divisionId = to.divisionId;
              isNew = false; }
public boolean needsLoading() {
              if (pm.needLoading(employee)) return true;
              else return false;
       }
}
```


Capa de integración
Almacén del dominio

```
public class EmployeeTO {
       public String id;
       public String lastName;
       public String firstName;
       public String ss;
       public float   salary;
       public String divisionId;
```


Capa de integración
Almacén del dominio

```
public EmployeeTO(String id, String lastName,
                    String firstName, String ss,
                    float salary, String divisionId ) {
              this.id = id;
              this.lastName = lastName;
              this.firstName = firstName;
              this.ss = ss;
              this.salary = salary;
              this.divisionId = divisionId;
       }
}
```

Capa de integración
Almacén del dominio

```
public class EmployeeStoreManager {
public void storeNew(EmployeeTO to) {
String sql = "Insert into Employee( id, last_name," +
             " first_name, ss, salary, division_id ) " +
             " values( '?', '?', '?', '?', '?', '?' )";
             . . .
       }

public void update(EmployeeTO to) {
String sql = "Update Employee set last_name = '?'," +
             " first_name = '?', salary = '?'," +
             " division_id = '?' where id = '?'";
             . . . }
```


Capa de integración
Almacén del dominio

```
public void delete(String empId) {
String sql = "Delete from Employee where id = '?'";
             . . .
       }

public EmployeeTO load(String empId) {
       . . .
       }
}
```


Capa de integración
Almacén del dominio

```
public class PersistenceManagerFactory {
static private PersistenceManagerFactory me = null;
public synchronized static PersistenceManagerFactory
getInstance() {
       if (me == null) {
                    me = new PersistenceManagerFactory();
       }
       return me;
       }
private PersistenceManagerFactory() { }
public PersistenceManager getPersistenceManager() {
              return new PersistenceManager();
       }
}
```


Capa de integración
Almacén del dominio

```
import javax.transaction.*;
import java.util.HashSet;
import java.util.Iterator;

public class PersistenceManager {
       HashSet stateManagers = new HashSet();
       TransactionManager tm;
       Transaction txn;

       public PersistenceManager() {
              tm = TransactionManager.getInstance();
              tm.register(this);
       }
```

Capa de integración
Almacén del dominio

```
public Persistable persistNew(Persistable o) {
       if (o instanceof Employee) {
              return setupEmployee(                    new
EmployeeStateDelegate((Employee)o), true);
              }
       return o;
       }

public Persistable persist(Persistable o) {
              // Must already be an EmployeeStateDelegate
       if (o instanceof Employee) {
       EmployeeStateDelegate esd =(EmployeeStateDelegate) o;
                                  return esd;       }
       return o; }
```


Capa de integración
Almacén del dominio

```
public void commit() throws SystemException,
NotSupportedException, HeuristicRollbackException,
RollbackException, HeuristicMixedException {
       if (txn == null) {
                     throw new SystemException(
                     Must call Transaction.begin() before" +
                                   " Transaction.commit()"); }
       Iterator i = stateManagers.iterator();
       while (i.hasNext()) {
              Object o = i.next();
              StateManager stateManager = (StateManager) o;
              stateManager.flush();
              }
              txn.commit();
              txn = null;   }
```


Capa de integración
Almacén del dominio

```
public void begin()throws SystemException,
NotSupportedException {
              txn = tm.getTransaction();
              txn.begin();
       }

public Employee getEmployee(String employeeId) {
       EmployeeStateDelegate esd =
              new EmployeeStateDelegate(employeeId);
              setupEmployee(esd, false);
              return esd;
       }
```


Capa de integración
Almacén del dominio

```
private EmployeeStateDelegate setupEmployee(
              EmployeeStateDelegate esd, boolean isNew)
{
       EmployeeStateManager stateManager =
       new EmployeeStateManager(this, esd, isNew);
       stateManagers.add(stateManager);
       esd.setStateManager(stateManager);
       return esd;
       }

public void setDirty(Persistable o) {
              // set dirty marker to true
       }
```

Capa de integración
Almacén del dominio

```
public void resetDirty(Persistable o) {
        // reset dirty marker to false
    }

public boolean isDirty(Persistable o) {
        // check if object is dirty
        return true;
    }

public boolean needLoading(Persistable o) {
        // check if needs to be loaded
        return true;
    } }
```


Capa de integración
Almacén del dominio

```
import javax.transaction.*;
import java.util.Iterator;
import java.util.LinkedList;
public class TransactionManager {
    static TransactionManager me = null;
    private LinkedList persistenceManagers = new
LinkedList();
    class PManager {
        Thread thread;
        PersistenceManager manager;
```


Capa de integración
Almacén del dominio

```
    PManager(Thread thread, PersistenceManager manager) {
        this.thread = thread;
        this.manager = manager;
    }
    boolean equals(Thread thread,
    PersistenceManager manager) {
        if (this.thread == thread &&
                this.manager == manager) {
            return true;
        }
        return false;
    }
}
```


Capa de integración
Almacén del dominio

```
public synchronized static TransactionManager getInstance()
{
        if (me == null) {
            me = new TransactionManager();
        }
        return me;
    }

private TransactionManager() { }

public Transaction getTransaction() {
        return new Transaction();
    }
```

Capa de integración
Almacén del dominio

```
public void register(PersistenceManager manager) {
            . . .  }

public void notifyCommit(Thread t)
                  throws SystemException,
                        HeuristicRollbackException,
                        NotSupportedException,
                        RollbackException,
                        HeuristicMixedException {
            Iterator i = persistenceManagers.iterator();
            while (i.hasNext()) {
                  . . .
                  pm.manager.commit();
            }     }   }
```


Capa de integración
Almacén del dominio

```
import javax.ejb.SessionContext;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.transaction.*;

public class Transaction {
      UserTransaction txn;

public void setSessionContext( SessionContext ctx ) {
            ctx.getUserTransaction();
      }
```


Capa de integración
Almacén del dominio

```
public Transaction() {
            InitialContext ic = null;
            try {
                  ic = new InitialContext();
      txn =
(UserTransaction)ic.lookup("java:comp/UserTransaction")
;
            } catch (NamingException e) {
            }     }

public void begin() throws SystemException,
                  NotSupportedException {
            txn.begin();  }
```


Capa de integración
Almacén del dominio

```
public void commit()
                  throws SystemException,
                        HeuristicRollbackException,
                        RollbackException,
                        HeuristicMixedException {
            txn.commit();
      }

public void rollback() throws SystemException {
            txn.rollback();
      }
}
```

Micro arquitectura trabajador web

- Propósito
  - Se desea que un sistema de workflow dirija a los usuarios a la página web de una aplicación J2EE para completar sus tareas
- También conocido como
  - Web worker micro-architecture

Micro arquitectura trabajador web

- Motivación
  - En la actualidad existen sistemas de workflow que ayudan a realizar tareas complejas en las que cooperan software y personas
  - Es posible que estos sistemas de workflow tengan que invocar servicios de aplicación existentes
  - Después, el control debería volver al sistema de workflow

Micro arquitectura trabajador web

- Contexto
  - Existe lógica de proceso negocio que es identificable y cambiable
  - Existe lógica de proceso de negocio sofisticada y que puede ser ejecutada como un subsistema diferente
  - Existen un elemento de trabajo de proceso de negocio centrado en el usuario

Micro arquitectura trabajador web

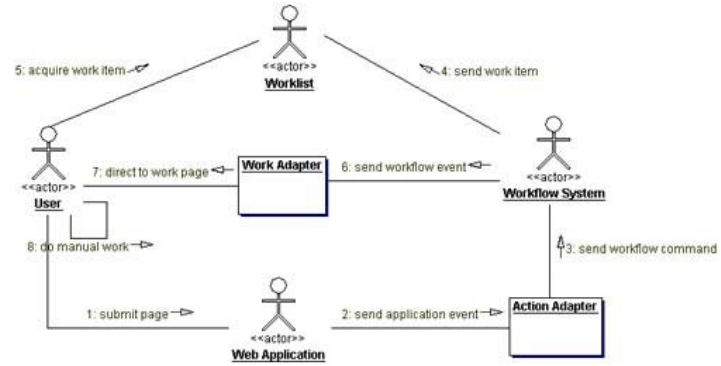
  - Las definiciones de workflow están representadas por actividades que suceden secuencialmente y/o en paralelo
  - La lógica de proceso necesita ser coordinada con usuarios
  - Se desea que el usuario adquiera el trabajo de lista de trabajo y finalice el trabajo vía aplicación web

Micro arquitectura trabajador web

- Solución
  - Utilizar el trabajador web para integrar usuarios, una aplicación web y un sistema de workflo

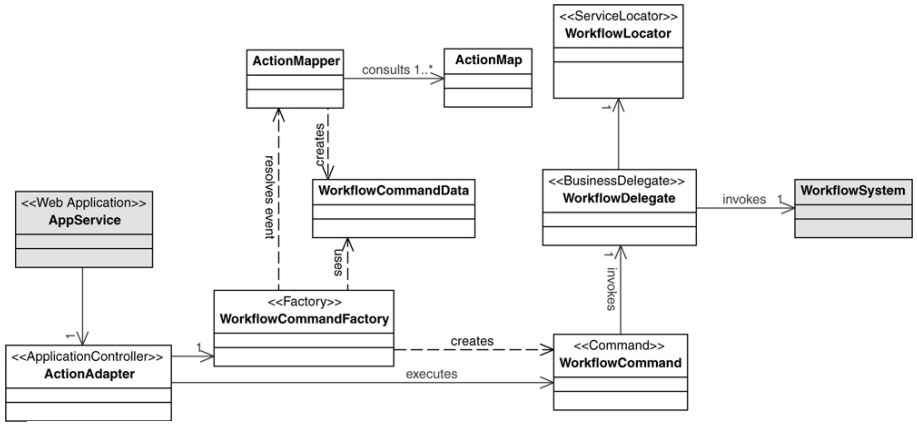
Micro arquitectura trabajador web

- Descripción



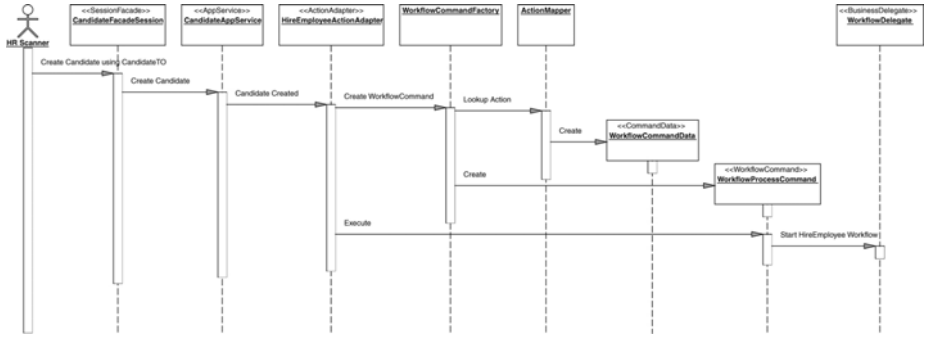
Interacción en la micro arquitectura

Micro arquitectura trabajador web



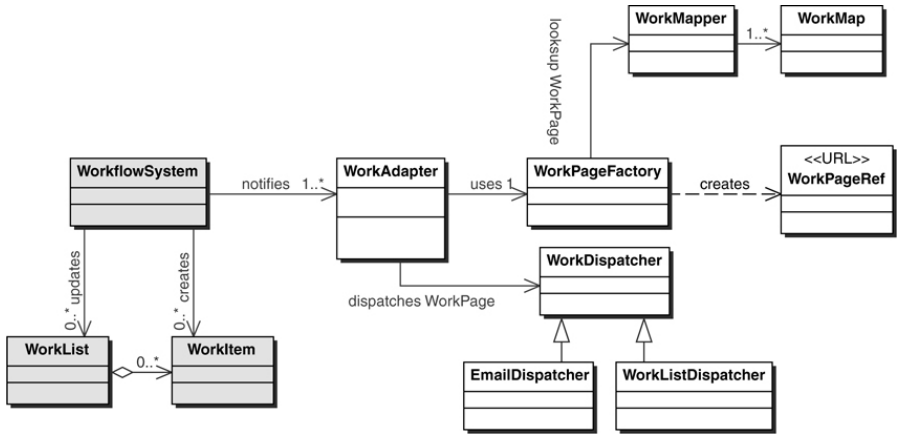
Estructura de comunicación aplicación a sistema workflow

Micro arquitectura trabajador web



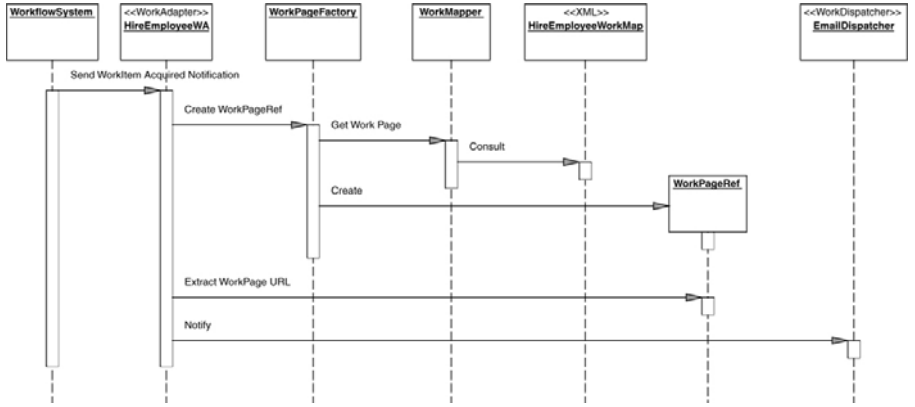
Ejemplo de invocación del sistema de workflow desde la aplicación

Micro arquitectura trabajador web



Estructura de comunicación del sistema workflow a aplicación

Micro arquitectura trabajador web



Ejemplo de invocación de aplicación desde el sistema de workflow

Micro arquitectura trabajador web

- Consecuencias
  - Ventajas
    - Separación de lógica de negocio y de procesamiento
    - Separación clara de preocupaciones
    - Alivia a la lógica de negocio de conocer la lógica de procesamiento
    - Permite al desarrollador escribir código de lógica de negocio sin verse afectado por cambios en el código de flujo de negocio

Micro arquitectura trabajador web

    - Asocia el rol de seguridad con lógica de procesamiento
    - Crea una separación de roles de seguridad entre lógica de negocio y lógica de proceso
  - Inconvenientes
    - Requiere conocimiento adicional y mantenimiento del sistema de workflow
    - Requiere conocimiento para la definición del proceso de workflow
    - Requiere código a medida para integrar con el sistema de workflow

Conclusiones

- Patrones arquitectónicos
- Unos son más prácticos que otros
- Front/application controller
- View helper
- Service to worker
- Business delegate
- Service locator

Conclusiones

- Application service
- Business object
- Transfer object
- DAO
- Domain store
- Problemas con concurrencia y persistencia