
**UNIVERSIDAD TÉCNICA DEL NORTE**

**FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS**

**CARRERA DE INGENIERÍA EN SISTEMAS COMPUTACIONALES**

**TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA EN SISTEMAS COMPUTACIONALES**

**TEMA:**

# "ESTUDIO DE PATRONES DE DISEÑO EN PLATAFORMA JAVA ENTERPRISE EDITION VERSIÓN 6 PARA EL DESARROLLO DE APLICACIONES WEB"

**AUTORA:**
Maricruz de Lourdes Acosta Yerovi

**DIRECTOR:**
Ing. José Luis Rodríguez

**Ibarra - Ecuador**
**2013**

## CERTIFICACIÓN

Por medio de la presente certifico:

Que la Srta. Maricruz de Lourdes Acosta Yerovi, egresada de la Carrera de Ingeniería en Sistemas Computacionales de la Facultad de Ingeniería en Ciencias Aplicadas de la Universidad Técnica del Norte, es la autora intelectual y material del trabajo de grado titulado **"ESTUDIO DE PATRONES DE DISEÑO EN PLATAFORMA JAVA ENTERPRISE EDITION VERSIÓN 6 PARA EL DESARROLLO DE APLICACIONES WEB"**, certificación que confiero por haber desempeñado las funciones de Director de Tesis durante el tiempo empleado en la elaboración y desarrollo del mencionado estudio.

Ing. José Luis Rodríguez
**DIRECTOR DE TESIS**


## UNIVERSIDAD TÉCNICA DEL NORTE

## CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE GRADO A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

Yo, MARICRUZ DE LOURDES ACOSTA YEROVI, con cédula de identidad Nro. 100204945-8, manifiesto mi voluntad de ceder a la Universidad Técnica del Norte los derechos patrimoniales consagrados en la Ley de propiedad Intelectual del Ecuador, artículos 4, 5 y 6, en calidad de autora de la obra o trabajo de grado denominado: **"ESTUDIO DE PATRONES DE DISEÑO EN PLATAFORMA JAVA ENTERPRISE EDITION VERSIÓN 6 PARA EL DESARROLLO DE APLICACIONES WEB"**, que ha sido desarrollado para optar por el título de Ingeniera en Sistemas Computacionales, en la Universidad Técnica del Norte , quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente. En mi condición de autor me reservo los derechos morales de la obra antes citada. En concordancia suscribo este documento en el momento que hago entrega del trabajo final en formato impreso y digital a la Biblioteca de la Universidad Técnica del Norte.

Ibarra, a los 24 días del mes de junio de 2013

Nombre: Acosta Yerovi Maricruz de Lourdes
Cédula: 100204945-8

**UNIVERSIDAD TÉCNICA DEL NORTE**

**BIBLIOTECA UNIVERSITARIA**

## AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

### 1. IDENTIFICACIÓN DE LA OBRA

La Universidad Técnica del Norte dentro del proyecto Repositorio Digital Institucional, determinó la necesidad de disponer de textos completos en formato digital con la finalidad de apoyar los procesos de investigación, docencia y extensión de la Universidad.

Por medio del presente documento dejo sentada mi voluntad de participar en este proyecto, para lo cual pongo a disposición la siguiente información:

| DATOS DE CONTACTO | | | |
|---|---|---|---|
| **CÈDULA DE IDENTIDAD:** | 100204945-8 | | |
| **APELLIDOS Y NOMBRES:** | ACOSTA YEROVI MARICRUZ DE LOURDES | | |
| **DIRECCIÓN:** | Sánchez y Cifuentes 3-43 y Rafael Troya | | |
| **EMAIL:** | jmary_86@hotmail.com / jcmary86@gmail.com | | |
| **TELÉFONO FIJO:** | 062607329 | **TELÉFONO MÓVIL:** | 0987809394 |

| DATOS DE LA OBRA | |
|---|---|
| **TÌTULO:** | "ESTUDIO DE PATRONES DE DISEÑO EN PLATAFORMA JAVA ENTERPRISE EDITION VERSIÓN 6 PARA EL DESARROLLO DE APLICACIONES WEB" |
| **AUTORA:** | MARICRUZ DE LOURDES ACOSTA YEROVI |
| **FECHA:** | 24 DE JUNIO DE 2013 |
| **PROGRAMA:** | PREGRADO |
| **TÍTULO POR EL QUE OPTA:** | INGENIERA EN SISTEMAS COMPUTACIONALES |
| **DIRECTOR:** | ING. JOSÉ LUIS RODRÍGUEZ |

## 2. AUTORIZACIÓN DE USO A FAVOR DE LA UNIVERSIDAD

Yo, MARICRUZ DE LOURDES ACOSTA YEROVI, con cédula de identidad Nro. 100204945-8, en calidad de autor y titular de los derechos patrimoniales de la obra o trabajo de grado descrito anteriormente, hago entrega del ejemplar respectivo en formato digital y autorizo a la Universidad Técnica del Norte, la publicación de la obra en el Repositorio Digital Institucional y uso del archivo digital en la Biblioteca de la Universidad con fines académicos, para ampliar la disponibilidad del material y como apoyo a la educación, investigación y extensión; en concordancia con la Ley de Educación Superior Artículo 144.

Ibarra, a los 24 días del mes de junio de 2013

Nombre: Acosta Yerovi Maricruz de Lourdes
Cédula: 100204945-8

## 3. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 24 días del mes de junio de 2013

**EL AUTOR:**

Nombre: Acosta Yerovi Maricruz de Lourdes
Cédula: 100204945-8

# DEDICATORIA

Este trabajo está dedicado con mucho cariño y gratitud a las personas más importantes en mi vida. A mis padres Lourdes y César, a mi abuelita Marujita y a mis hermanas Ivonne y Tatiana. Los quiero mucho.

*Mary*

# AGRADECIMIENTO

A Dios, por iluminar mi camino y haberme dado de su infinita sabiduría para finalizar con éxito esta etapa de mi vida.

A mis padres, mi abuelita y hermanas, por todo el apoyo y el amor incondicional que siempre me han brindado.

Al Ingeniero José Luis Rodríguez, Director del Proyecto, y al Ingeniero Jorge Caraguay, quienes con sus conocimientos y consejos supieron guiarme e hicieron posible la culminación de mi trabajo.

A los Ingenieros Daniel Jaramillo, Diego Ortiz, Fernando Garrido y Mauricio Rea, mil gracias por su amistad, ayuda y por compartir sus conocimientos y experiencias.

*Mary*

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# RESUMEN

En la búsqueda constante por mejorar la calidad y el diseño del software, técnicas y tecnologías han evolucionado. Se han desarrollado diversas estrategias y habilidades, varias son producto de mejoras realizadas sobre estrategias anteriores. Los patrones de diseño probablemente son las estrategias más versátiles, consisten en la reutilización de soluciones, no solamente de código fuente, sino que constituyen soluciones integrales a problemas repetitivos en el desarrollo del software.

Es importante señalar que los patrones de diseño JEE, recopilan un conjunto de buenas prácticas que se han venido desarrollando en los últimos años para el desarrollo de sistemas web. La presencia de patrones de diseño conduce a soluciones estándares, fácilmente comprensibles y mantenibles por parte de los desarrolladores.

El enfoque principal de este trabajo es el conocimiento de los patrones de diseño JEE 6, ya que a pesar de los beneficios que su implementación genera, son tratados a menudo como patrones J2EE, obstruyendo la funcionalidad de los patrones de diseño y de la plataforma en sí.

# SUMMARY

In the ongoing quest to improve the quality and design of software, techniques and technologies have evolved. Various strategies have been developed, several improvements are the result of previous strategies. Design patterns are probably the most versatile strategies consist reuse solutions, not only of source code, but provide solutions to recurring problems in software development.

Importantly JEE design patterns, collected a set of best practices that have been developed in recent years for the development of web systems. The presence of design patterns leads to standard solutions, easily understandable and maintainable by developers.

The main focus of this work is the knowledge of design patterns JEE 6, and that despite the benefits it generates implementation, are often treated as J2EE patterns, blocking the functionality of the design patterns and platform.

# CAPÍTULO I

# 1 Introducción

## 1.1 Motivación

Desde los años 90s, cuando Internet empieza a propagarse mundialmente, se inicia la búsqueda de tácticas y destrezas con el fin de aprovechar las nuevas tecnologías y brindar soluciones que se adapten al medio. Esta propagación, hizo que los ingenieros y desarrolladores de software paulatinamente vayan dejando atrás algunas arquitecturas que, aunque continúan siendo vigentes, tienden a desaparecer con el paso de los años. El problema no radica en que las tecnologías cambien, sino en la capacidad que el desarrollador tenga para adaptarse a ellas. Precisamente, éste ha sido uno de los obstáculos principales en el desarrollo de software para la Web.

En la actualidad, se dispone de diferentes alternativas para la implementación de aplicaciones Web como, diferentes lenguajes de programación, arquitecturas y técnicas. Además, se cuenta con la funcionalidad que proveen los diversos patrones de diseño web existentes. No obstante, a la hora de desarrollar un proyecto Web, surgen dudas acerca de qué alternativa utilizar, ya que cada una posee un alcance, ventajas y desventajas, y de la elección realizada, depende el éxito o fracaso de un proyecto.

Este trabajo pretende ser una guía que permita elegir el o los patrones de diseño de la plataforma Java EE 6 más adecuados para la implementación de aplicaciones web, tomando en cuenta las características principales que cada uno de ellos tiene y los beneficios que ofrecen.

## 1.2 Problema

A pesar de la existencia y amplio uso de los patrones de diseño, aún existe cierto grado de desconocimiento a cerca de la existencia y beneficios de la utilización de patrones de diseño de Java Enterprise Edition 6 para el desarrollo de aplicaciones Web. Un considerable número de desarrolladores utilizan patrones de la plataforma J2EE en aplicaciones JEE, lo cual imposibilita el uso adecuado de la plataforma y de las mejoras que ésta ofrece para el diseño y producción de sistemas, incrementando de cierta forma, el tiempo de desarrollo y entrega de los mismos.

Este documento pretende ser una fuente de consulta que provea información clara y sintetizada para que los programadores interesados en conocer nuevas técnicas y mejorar su productividad, hagan uso de los diferentes patrones de diseño JEE 6 en la construcción de aplicaciones flexibles y fáciles de implementar.

## 1.3 Objetivos

### 1.3.1 Objetivo General.

Realizar un estudio de los patrones de diseño de la plataforma JEE 6 que intervienen en las diferentes capas de las aplicaciones Web, con el fin de conocer su utilidad, funcionalidad y empleo en el desarrollo de dichas aplicaciones.

### 1.3.2 Objetivos Específicos.

- Estudiar los patrones de diseño JEE 6 y su aplicación.
- Determinar los beneficios de la utilización de patrones de diseño JEE 6.
- Desarrollar un sistema hotelero.

- Definir los patrones de diseño que serán utilizados en el desarrollo del sistema hotelero.
- Documentar el desarrollo del sistema utilizando metodología RUP.
- Realizar pruebas del sistema para verificar su correcto funcionamiento.
- Capacitar a los usuarios en la utilización del sistema.

## 1.4 Alcance

El tema propuesto contempla el estudio de patrones de diseño de Java Enterprise Edition versión 6 para aplicaciones Web. Se adoptará la arquitectura Modelo Vista Controlador (MVC) para el desarrollo del sistema hotelero y los patrones de diseño que se emplearán, serán seleccionados a medida que se vaya realizando la investigación.

## 1.5 Justificación

Pueden surgir varias interrogantes al momento de iniciar el desarrollo de un proyecto Web. Estos proyectos suelen delimitarse con tiempos críticos de entrega y deben ejecutarse disminuyendo gastos. Además, deben ser sistemas ligeros en consumo de recursos, escalables y capaces de intercambiar información. La mantenibilidad también es fundamental en este tipo de proyectos.

Este trabajo está enfocado en las buenas prácticas de desarrollo de aplicaciones Web y, al existir cierto grado de desconocimiento sobre el tema, se analizará cada patrón de diseño de la plataforma Java EE versión 6; por lo tanto, la realización de este estudio busca que el lector adquiera un determinado nivel de conocimiento de los patrones de diseño y su posible implementación, pretende dar a conocer los beneficios de la utilización de patrones de diseño JEE 6 y, busca ser una guía para la elección de los patrones adecuados.

Disponer de esta investigación es importante, ya que el uso apropiado de patrones de diseño JEE 6 en la construcción de aplicaciones, permite la optimización de código, reduciendo el tiempo de entrega de los sistemas y haciendo más fácil el mantenimiento de los mismos.

## 1.6 Antecedentes

Los patrones de diseño se originaron como conceptos de arquitectura civil por el arquitecto Christopher Alexander (1977). Luego en el contexto de OOPSLA 87[1], Kent Beck y Ward Cunningham (1987) empezaron a aplicar el concepto de patrones en la informática.

En 1991, Erich Gamma y Richard Helm trabajaron en la elaboración de su primer catálogo, popularizado tras la publicación del libro *Design Patterns* (1994). Actualmente existen diferentes tipos de patrones que buscan solucionar la mayoría de problemas que se generan en el desarrollo del software.

Posteriormente Martin Fowler et al., publican el libro *Patterns of Enterprise Application Architecture* (2002), en el cual, las aplicación empresariales son divididas en diferentes capas y se enfatiza el uso de los patrones relacionados con mappings entre objetos y bases de datos relacionales. A continuación, Deepak Alur, John Crupi y Dan Malks publican la segunda edición de su libro, divulgado en el 2001, enfocado en las buenas prácticas y estrategias de diseño para aplicaciones empresariales, *Core J2EE Patterns* (2003). En este trabajo se estudian 21 patrones de diseño con ejemplos de código y puntos importantes para la refactorización de este tipo de aplicaciones utilizando dichos patrones.

[1] **OOPSLA,** Conferencia sobre Sistemas de Programación Orientada a Objetos, Idiomas, y Aplicaciones (Meyrowitz, Norman K.).

William Crawford y Jonathan Kaplan hicieron público su trabajo *J2EE Design Patterns* (2003). Este libro se direcciona al estudio de patrones de diseño para aplicaciones empresariales y abarca temas como extensibilidad, mantenimiento, escalabilidad, modelado de datos, transacciones e interoperabilidad.

En última instancia, los diseños utilicen patrones o no, deben ser traducidos a código, por lo que se publica el libro *Real World Java EE Patterns – Rethinking Best Practices* (2009) escrito por Adam Bien, en el cual se hace un replanteamiento de los patrones de diseño que ahora son utilizados en aplicaciones JEE. Debido a la gran acogida que tuvo su lanzamiento, el libro *Real World Java EE Night Hacks* (2011), es publicado como complemento al libro anterior, en él se refleja la utilización de los patrones en aplicaciones web.

En la actualidad, existen al menos tres grandes tecnologías asociadas a la programación Web: ASP.NET[2], JEE [3] y PHP[4].

[2] **ASP.NET,** framework para aplicaciones web desarrollado y comercializado por Microsoft (WWW02).

[3] **JEE,** plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java (WWW03).

[4] **PHP,** lenguaje de programación originalmente diseñado para el desarrollo web de contenido dinámico (WWW04).

Java Enterprise Edition existe desde el año 1998 como iniciativa de un grupo de ingenieros de Sun Microsystems[5], y en más de 10 años de desarrollo se ha convertido en una de las arquitecturas más robustas para la construcción de sistemas de información. JEE fue pionera en la elaboración de patrones de diseño, publicando el 8 de Marzo del 2001, su propio catálogo de patrones que tuvo gran aceptación, y, posteriormente difunde los patrones *BluePrints*[6] e incorpora en los sistemas conceptos vitales como escalabilidad y rendimiento.

Se ha optado por el análisis de patrones de diseño JEE, porque las principales fuentes de patrones Web utilizan esta plataforma de implementación, aunque en la práctica, los patrones son suficientemente independientes de la plataforma.

[5] **Sun Microsystems,** empresa informática dedicada a vender estaciones de trabajo, servidores, componentes informáticos, software y servicios informáticos; fue adquirida en el 2009 por Oracle Corporation (WWW05).

[6] **BluePrints,** catálogo de patrones basado en el libro Core J2EE Patterns, utilizados en el contexto de J2EE (WWW06).

# CAPÍTULO II

# 2 JEE (Java Enterprise Edition)

## 2.1 Introducción

Java Enterprise Edition anteriormente conocida como J2EE (Java Second Enterprise Edition) fue desarrollada por Sun Microsystems, iniciando con la liberación de J2EE 1.3, la especificación fue desarrollada bajo el Java Community Process[7].

J2EE 1.4 fue liberado en diciembre de 2002, posteriormente Java EE 5 fue desarrollada bajo el JSR 244, su liberación se produce el 11 de mayo de 2006; y finalmente Java EE 6 que fue desarrollada bajo el JSR 316 con su liberación el 10 de diciembre de 2009 (WWW09).

## 2.2 Definición

Según el sitio web oficial de Oracle:

Java Enterprise Edition es una arquitectura tecnológica que se basa en el sólido fundamento de Java Plataform, Standard Edition (J2SE) y es el estándar de la industria para el desarrollo de aplicaciones empresariales, aplicaciones orientadas a servicios (SOA) y aplicaciones web de próxima generación (WWW10).

El sitio web oficial de Oracle define a JEE versión 6 como:

[7] **Java Community Process,** o Proceso de la Comunidad Java (JCP), establecido en 1998, es un proceso formalizado que permite a las partes interesadas involucrarse en la definición de futuras versiones y características de la plataforma Java. El proceso JCP conlleva el uso de Java Specification Request (JSR), documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java (WWW07).

Java Plataform, Enterprise Edition (Java EE) 6 es el estándar de la industria para la informática empresarial de Java. Permite utilizar el nuevo y ligero Java EE 6 Web Profile para crear aplicaciones Web de próxima generación, y todo el poder de Java EE 6 Plataform para aplicaciones empresariales. Los desarrolladores se beneficiarán de las mejoras de la productividad con más anotaciones, más POJO's[8], la simplificación de empaquetado, y menos configuración de XML (WWW11).

Las mejoras que incluye JEE 6 son: la inclusión de perfiles, los cuales permiten utilizar ciertos fragmentos de toda la especificación (Perfil Completo y Perfil Web). Actualización de JAX-RS y JAX-WS, JPA 2.0 con nuevas librerías, Servlets[9] 3.0 los cuales incluyen varios POJO's, EJB (Enterprise Java Beans) 3.1; entre los más importantes.

## 2.3 Arquitectura aplicaciones JEE

El modelo JEE comienza con el lenguaje de programación Java y la JVM, y, soporta aplicaciones que implementan servicios empresariales. Los requerimientos de los clientes son tratados en la capa media del modelo, la cual corre en un hardware dedicado, tiene acceso a los recursos empresariales y divide el trabajo en dos partes: la lógica del negocio y la presentación.

[8] **Plain Old Java Object (POJO),** es una sigla creada en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial (WWW12).

[9] **Servlets**, objetos que corren dentro y fuera del contexto de un contenedor de servlets y extienden su funcionalidad. Su uso más común es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web (WWW13).

El modelo de aplicaciones de Java EE define una arquitectura para la implementación de servicios como aplicaciones multinivel, que ofrecen escalabilidad, accesibilidad y facilidad de gestión, características necesarias en aplicaciones de nivel empresarial (OracleCorporation, pág. 39).

En este tema se observan los componentes más importantes de la arquitectura Java EE y los servicios que provee, los cuales pueden ser empleados en las diferentes aplicaciones.

### 2.3.1 Modelo de capas.

La plataforma Java EE adopta un modelo distribuido de aplicaciones multinivel para aplicaciones empresariales, donde la lógica de la aplicación se divide en componentes según su función. Utiliza una lógica de programación desarrollada en niveles o capas, que permite encapsular o separar elementos de las aplicaciones en partes definidas, para simplificar el desarrollo y esclarecer las responsabilidades de cada uno de los componentes de un aplicativo.

Inicialmente, en un entorno sin capas, el desarrollo de aplicaciones entremezclaba todas las tareas en el lado servidor, como puede verse en la figura 1[10].

[10] Figura obtenida del Manual de Desarrollo Básico de Aplicaciones en la Plataforma J2EE (Miguel Abarca C., pág. 9)

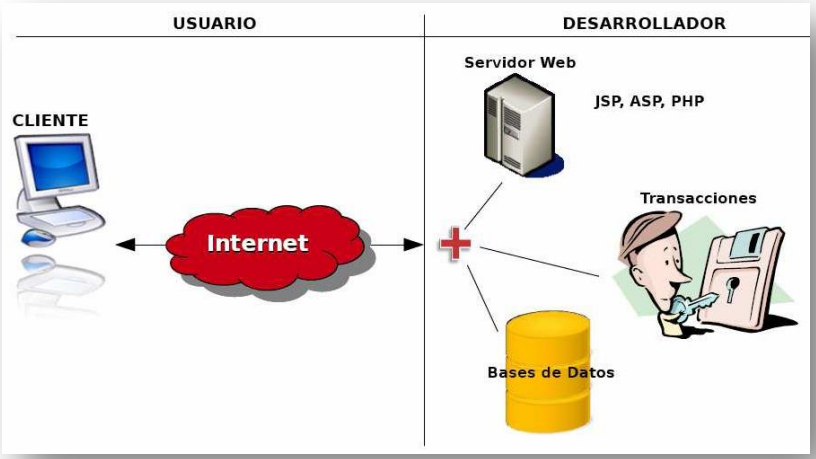


***Figura 1: Modelo de aplicaciones sin capas***

Las aplicaciones Java EE se basan en el modelo de aplicaciones en capas en donde, la capa Web contiene el Servidor Web y las transacciones se realizan en la capa de Negocio (Óscar Belmonte, 2012); con lo que las operaciones de negocio y generación de HTML dinámico que se envían al navegador se encuentran separadas, permitiendo un mejor control de los elementos que intervendrán en el desarrollo de aplicaciones web.

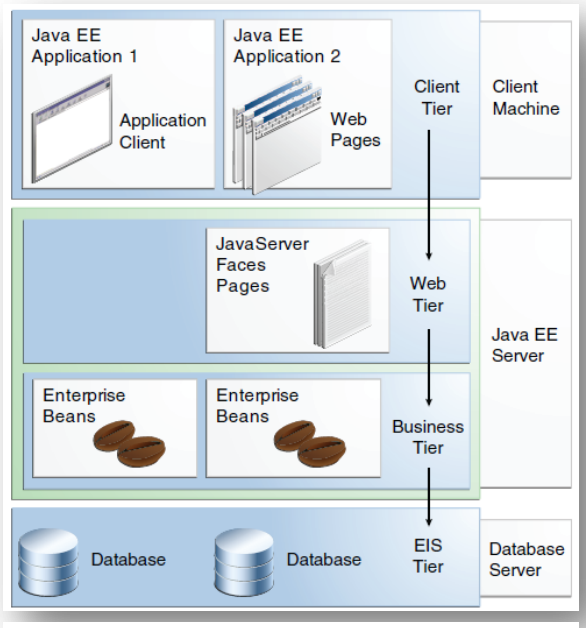


***Figura 2: Aplicaciones Multicapa***

[11] Figura obtenida del Tutorial de JEE 6 (OracleCorporation, pág. 41).

La figura 2, muestra dos aplicaciones Java EE multinivel divididas en las capas descritas a continuación:

- Los componentes de la Capa Cliente se ejecutan en la máquina del cliente.
- Los componentes de la Capa Web se ejecutan en el servidor Java EE.
- Los componentes de la Capa del Negocio se ejecutan en el servidor Java EE.
- La Capa del Sistema de Información Empresarial (EIS) ejecuta el software en el servidor EIS.

Aunque una aplicación Java EE puede tener tres o cuatro niveles, es considerada generalmente como una aplicación de tres niveles porque están distribuidas en tres lugares: máquinas del cliente, la máquina del servidor Java EE y la base de datos.

### 2.3.2 Componentes Java EE.

Las aplicaciones Java EE están formadas por componentes. Un componente Java EE es una unidad auto-contenida de software funcional, que se ensambla en una aplicación Java EE con sus clases relacionadas y archivos, y, se comunica con otros componentes (OracleCorporation, pág. 42).

La especificación Java EE define los siguientes componentes:

- Clientes JEE (cliente Web o aplicación del cliente) y applets, son componentes que se ejecutan en el cliente.
- Java Servlets, JavaServer Faces y componentes de la tecnología JavaServer Pages (JSP), son componentes web que se ejecutan en el servidor.

- Enterprise JavaBeans (EJB), Enterprise Beans, son componentes de negocio que se ejecutan en el servidor.

### 2.3.3 Tipos de contenedores JEE.

Los contenedores son la interfaz entre un componente y la funcionalidad de una plataforma específica de bajo nivel que soporta ese componente. Antes de que un componente web, bean enterprise o aplicación del cliente sea ejecutado, debe ser ensamblado en un módulo Java EE y desplegado en su contenedor (OracleCorporation, pág. 47).

Entonces un contenedor puede considerarse como un entorno de ejecución para los componentes.

El contenedor gestiona también servicios no configurables como beans Enterprise, ciclos de vida de los servlets, persistencia de datos y el acceso a las APIs de la plataforma Java EE.

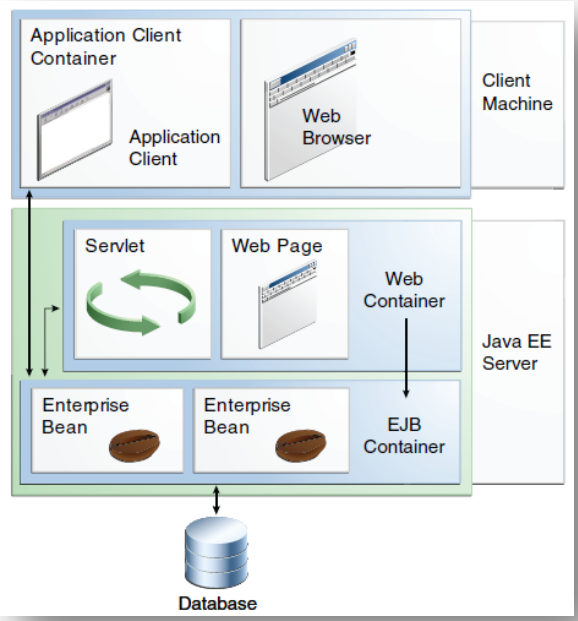


***Figura 3: Contenedores y Servidor Java EE***

En la figura 3[12], se pueden observar los siguientes contenedores:

- Servidor Java EE: Un servidor Java EE proporciona contenedores EJB y Web.
- Contenedor Enterprise JavaBeans (EJB): Administra la ejecución de beans empresariales para aplicaciones Java EE. Los Enterprise Beans y su contenedor se ejecutan en el servidor Java EE.
- Contenedor Web: Maneja la ejecución de páginas web, servlets, y algunos componentes EJB de aplicaciones Java EE. Los componentes Web y su contenedor se ejecutan en el servidor Java EE.

    - Contenedor de aplicaciones cliente: Gestiona la ejecución de los componentes de la aplicación cliente, esta aplicación y su contenedor se ejecutan en el cliente.
    - Contenedor de Applet: Controla la ejecución de applets. Consiste en un navegador web y Java Plug-in que se ejecutan juntos en el cliente (OracleCorporation, pág. 49).

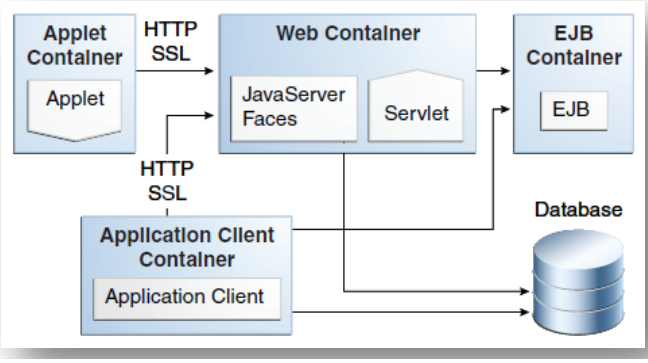


***Figura 4: Relaciones entre los contenedores Java EE***

[12] Figura obtenida del Tutorial de JEE 6 (OracleCorporation, pág. 48).

[13] Figura obtenida del Tutorial de JEE 6 (OracleCorporation, pág. 55).

### 2.3.4 Características y servicios de Java EE.

El objetivo fundamental de la plataforma Java EE 6 es simplificar el desarrollo, al proporcionar una base común para los distintos tipos de componentes de la plataforma Java EE. Los desarrolladores se benefician de las mejoras de productividad con más anotaciones y menos configuración XML, más POJOs y empaquetado simplificado. La plataforma Java EE 6 incluye las siguientes características nuevas:

- Perfiles: son configuraciones dirigidas a tipos específicos de aplicaciones. La plataforma Java EE 6 introduce un perfil web ligero dirigido a aplicaciones web de última generación, así como un perfil completo que contiene todas las tecnologías Java EE y ofrece toda la potencia de la plataforma Java EE 6 para aplicaciones empresariales.
- Nuevas tecnologías, entre ellas:
  - API de Java para servicios web RESTFUL (JAX-RS).
  - Beans gestionados.
  - Contextos e Inyección de Dependencia para la plataforma Java EE, conocida como CDI.
  - Inyección de Dependencia para Java.
    - Bean Validation
    - Java Authentication Service Provider Interface for Containers (JASPIC).
  - Nuevas características para componentes Enterprise JavaBeans.
  - Nuevas características para servlets.
  - Nuevas características para componentes JavaServer Faces.

A continuación se describen brevemente algunos servicios que provee Java EE y que se pueden emplear en cualquier servidor de aplicaciones que siga este estándar.

- *Java Transaction API (JTA)*: Es la API para transacciones en Java que define las interfaces entre el manejador de transacciones y las partes involucradas en un sistema de transacciones distribuidas: el manejador de recursos, el servidor de aplicaciones y las aplicaciones transaccionales.
- *Java Persistente API (JPA)*: API de persistencia, maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones Standard (Java SE) y Enterprise.
- *Java Message Service (JMS)*: El servicio de mensajes Java, permite a los componentes crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona.
- *Java Naming Direct Interface (JNDI)*: La Interfaz de Nombrado y Directorio Java es una API que permite a los clientes descubrir y buscar, objetos y datos, a través de un nombre.
- *JavaMail*: Facilita el envío y recepción de mails desde código java.
- *Java API for XML Processing (JAXP)*: Soporte para el manejo y tratamiento de archivos XML.
- *Arquitectura Java EE Connector (JCA)*: Admite el acceso a sistemas de información empresarial por medio del desarrollo de un pluggin, similar a los conectores JDBC. Permite la conexión a sistemas heredados, incluyendo bases de datos.
- *Java Autentication and Authorization Service (JAAS)*: Provee servicios de seguridad de autenticación y autorización.

- *Servicios Web (JAX-WS)*: Es una API de Java para la creación de servicios web, utiliza anotaciones para simplificar el desarrollo y despliegue de los clientes y puntos finales de servicios web.

Además, provee servicios para manejo de protocolos HTTP/HTTPS, IIOP/RMI, JDBC. En la figura[14] 5 se exponen las APIs de los Contenedores Web, EJB y Aplicación Cliente de Java EE 6.

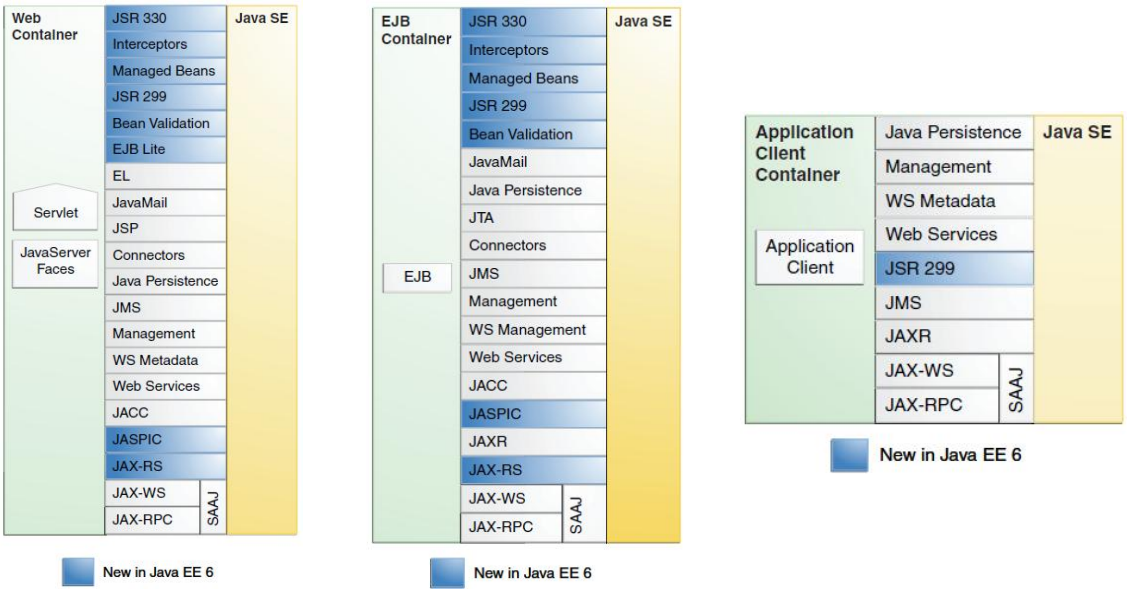


***Figura 5: APIs de la Plataforma Java EE 6***

[14] Figura obtenida del Tutorial de JEE 6 (OracleCorporation, págs. 56-58).

### 2.3.5 Empaquetado de aplicaciones.

Una aplicación Java EE es entregada en un archivo JAR[15], un archivo WAR[16] o un archivo EAR. Un WAR o EAR es un archivo JAR estándar con una extensión .war ó .ear. El uso de archivos JAR, WAR, EAR y módulos, hace posible ensamblar diferentes aplicaciones Java EE sin necesitar código adicional, sino que únicamente se empaquetan varios módulos Java EE en archivos JAR, WAR o EAR.

Un desarrollador de software realiza las siguientes tareas para entregar un archivo EAR que contiene la aplicación Java EE[17]:

- Ensambla los archivos JAR y WAR creados en fases anteriores en un archivo EAR de la aplicación Java EE.
- Opcionalmente, especifica el descriptor de despliegue de la aplicación Java EE.
- Verifica que el contenido del archivo EAR esté adecuadamente formado y cumpla con la especificación Java EE.

[15] **Archivo JAR,** permite ejecutar aplicaciones escritas en el lenguaje Java y están comprimidos con el formato ZIP (WWW16).

[16] **Archivo WAR,** es un archivo JAR utilizado para distribuir una colección de JavaServer Pages, servlets, clases Java, archivos XML, librerías de tags y páginas web estáticas que juntos constituyen una aplicación web (WWW17).

[17] Información obtenida del Tutorial de JEE 6 (OracleCorporation, págs. 54).

## 2.4 Tecnologías Java EE

En esta sección se expone un breve resumen de las tecnologías requeridas por la plataforma Java EE.

### 2.4.1 Tecnología Java Server Faces.

La tecnología JavaServer Faces (JSF) es un framework de interfaz de usuario que permite la creación de aplicaciones web. Los componentes principales de esta tecnología son:

- Un marco de componentes de interfaz de usuario.
- Un modelo flexible para renderizar componentes en distintos tipos de HTML o diferentes lenguajes de marcado y tecnologías. Un objeto Renderer convierte los datos almacenados en un objeto del modelo a tipos que pueden ser representados en una vista.
- Un RenderKit estándar para la generación de marcado HTML/4.01.

Esta funcionalidad está disponible mediante APIs estándar de Java y XML basados en archivos de configuración. Las nuevas características de JSF en la plataforma Java EE 6 son[18]:

- La capacidad de utilizar anotaciones en lugar de un archivo de configuración para especificar beans administrados y otros componentes.
- Facelets, una tecnología que reemplaza a JavaServer Pages (JSP) usando archivos XHTML.
- Ajax soporte.

[18] Información obtenida del Tutorial de JEE 6 (OracleCorporation, pág. 60).

- Componentes compuestos.
- Navegación implícita.

La plataforma Java EE 6 requiere el uso de JSF 2.0 y Expression Language 2.2.

### 2.4.2 Tecnología Enterprise Java Bean.

La tecnología EJB permite el desarrollo rápido y simplificado de aplicaciones distribuidas, transaccionales, seguras y portátiles basadas en la tecnología Java.

Un componente Enterprise JavaBeans (EJB) o enterprise bean, es el cuerpo del código que contiene campos y métodos para implementar módulos de lógica de negocio. Se puede pensar en un bean empresarial como un elemento que puede ser utilizado solo o con otros enterprise beans para ejecutar la lógica del negocio en el servidor Java EE.

En la especificación 3.0, los EJBs son POJOs con algunas funcionalidades especiales implícitas, que se activan en tiempo de ejecución cuando son ejecutados en un contenedor EJB.

El objetivo de los EJB's es proporcionar a los desarrolladores un modelo que permita abstraerse de los problemas generales de las aplicaciones empresariales (concurrencia, transacciones, persistencia, seguridad) para centrarse en el desarrollo de la lógica del negocio (WWW18).

### *2.4.2.1 Tipos de Enterprise Java Beans.*

Existen tres tipos de Enterprise Java Beans (EJB's):

- EJB's de entidad.
- EJB's de sesión.
- EJB's dirigidos por mensaje.

2.4.2.1.1 *EJB's de entidad (entity ejb's).*

Encapsulan los objetos del lado del servidor que almacenan los datos. Los EJB de entidad presentan la característica fundamental de la persistencia:

- Persistencia gestionada por el contenedor: Es el encargado de almacenar y recuperar los datos a través del Mapeo Objeto Relacional.

- Persistencia gestionada por el bean: El propio objeto se encarga de almacenar y recuperar los datos, por lo tanto el desarrollador es el encargado de implementar la persistencia.

*2.4.2.1.2 EJB's de sesión (session ejb's).*

Gestionan el flujo de la información en el servidor, sirven a los clientes como acceso a los servicios proporcionados por los otros componentes que se encuentran en el servidor.

- Con sesión (Statefull): Son objetos distribuidos que poseen un estado. El estado no es persistente, pero el acceso al bean se limita a un cliente.
- Sin sesión (Stateless): Son objetos distribuidos que carecen de un estado, permitiendo el acceso concurrente.

*2.4.2.1.3 EJB's dirigidos por mensaje (message-driven ejb's).*

Son beans con funcionamiento asíncrono. Utilizan el Sistema de Mensajes de Java (JMS), estos se suscriben a un tema o cola y se activan al recibir un mensaje dirigido a dicho tema o cola.

Las nuevas características del enterprise bean, en la plataforma Java EE 6, son los siguientes:

- Capacidad de empaquetar enterprise beans locales en un archivo WAR.
- Beans de sesión Singleton, que proporcionan un fácil acceso al estado compartido.
- Un subconjunto ligero de la funcionalidad de Enterprise JavaBeans (EJB Lite) que puede ser proporcionado dentro de Perfiles de Java EE, como el Perfil Web de Java EE.

La plataforma Java EE 6 requiere Enterprise JavaBeans 3.1 e interceptores 1.1.

### 2.4.3 Tecnología Java Servlet.

La tecnología Java Servlet permite definir clases específicas del servlet HTTP. Una clase servlet amplía las capacidades de los servidores que alojan aplicaciones accesibles, a través de un modelo de programación de petición-respuesta. Aunque los servlets pueden responder a cualquier tipo de petición, son empleados generalmente para extender las aplicaciones alojadas en servidores web (OracleCorporation, pág. 59).

En la plataforma Java EE 6, las nuevas características de esta tecnología son las siguientes:

- Soporte para anotaciones.
- Soporte asíncrono.
- Facilidad de configuración.
- Las mejoras de las APIs existentes.

La plataforma Java EE 6 requiere la implementación de Servlet 3.0.

### 2.4.4 Tecnología Javaserver Pages.

La tecnología JavaServer Pages (JSP), permite poner trozos de código servlet directamente en un documento de texto. Una página JSP es un documento que contiene dos tipos de texto (OracleCorporation, pág. 60):

- Datos estáticos, que pueden ser expresados en cualquier formato de texto como HTML o XML.
- Elementos JSP, que determinan cómo se construye la página de contenido dinámico.

La plataforma Java EE 6 requiere el uso de JavaServer Pages 2.2 y JSTL 1.2.

## 2.5 Herramientas de desarrollo del aplicativo

Con el fin poner en práctica la utilización de patrones de diseño, de desarrolló un sistema hotelero con la aplicación de las siguientes herramientas:

### 2.5.1 Entorno de desarrollo NetBeans.


***Figura 6: Logotipo del IDE NetBeans***

El IDE NetBeans[19] es un entorno de desarrollo integrado, es decir una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans que es un producto libre y gratuito sin restricciones de uso (WWW20).

Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, colaboración entre varias personas, creación de aplicaciones compatibles con teléfonos móviles, resaltado de sintaxis; en este IDE, se encontrará la solución más completa para programar en Java (WWW21).

Entre las características de la plataforma están:

- Administración de las interfaces de usuario.
- Administración de las configuraciones del usuario.
- Administración del almacenamiento.
- Administración de ventanas.
- Framework basado en asistentes (diálogo paso a paso).

[19] Figura tomada de la página de NetBeans.

### 2.5.2 Gestor de base de datos PostgreSql.

***Figura 7: Logotipo de PostgreSQL***

PostgreSQL[20] es un sistema gestor de base de datos (sgbd) relacional orientada a objetos y libre, publicado bajo licencia BSD (WWW24).

Entre sus principales características se encuentran[21]:

- Corre en los principales sistemas operativos: Linux, Unix, Mac OS, Windows.
- Documentación muy bien organizada, pública y libre, con comentarios de los propios usuarios.
- Comunidades muy activas, varias comunidades en castellano.
- Altamente adaptable a las necesidades del cliente.
- Soporte nativo para los lenguajes más populares del medio: PHP, C, C++, Perl, Python.
- Drivers: Odbc, Jdbc, .Net.
- Soporte de todas las características de una base de datos profesional (triggers, procedimientos, funciones, secuencias, relaciones, reglas, tipos de datos definidos por usuarios, vistas, etc.)

[20] Figura tomada de la página de PostreSql (WWW22).

[21] Información tomada del manual Introducción a PostgreSql (Quiñones).

- Soporte de protocolo de comunicación encriptado por SSL.
- Extensiones para alta disponibilidad, nuevos tipos de índices, datos espaciales, minería de datos, etc.
- Utilidades para análisis y optimización de Querys.

### 2.5.3 Servidor de aplicaciones GlassFish.

***Figura 8: Logotipo del Servidor GlassFish***

GlassFish[22] es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. La versión comercial es denominada Oracle GlassFish Enterprise Server. Es gratuito y de código libre, se distribuye bajo un licenciamiento dual a través de la licencia CDDL[23] y la GPL[24] de GNU[25] (WWW26).

Entre las características más importantes de GlassFish se destacan su velocidad, alta escalabilidad, manejo centralizado de clúster e instancias, bajo consumo de memoria, interoperabilidad con .NET 3 y un excelente panel de administración (WWW27).

[22] Figura tomada de la página de GlassFish (WWW25).

[23] **Licencia Común de Desarrollo y Distribución (CDDL),** Es una licencia de código abierto que proporciona una licencia de patente explícita para el código publicado (WWW28).

[24] **La Licencia Pública General de GNU (GPL de GNU),** se usa para la mayoría de los programas de GNU y para más de la mitad de los paquetes de software libre. La última versión es la 3 (WWW29).

[25] **GNU,** es un sistema operativo similar a Unix que es software libre y respeta su libertad (WWW30).

### 2.5.4 Framework JSF.


***Figura 9: Logotipo de JSF***

JavaServer Faces (JSF[26]) es una tecnología y framework para aplicaciones web, que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. Hace uso de JavaServer Pages como tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías como XUL (WWW31).

Sus objetivos son[27]:

- Definir un conjunto simple de clases Java para componentes de la interfaz de usuario, estado de los componentes y eventos de entrada.
- Proporcionar un conjunto de componentes para la interfaz de usuario, incluyendo los elementos estándares de HTML para representar un formulario. Estos componentes se obtendrán de un conjunto básico de clases base que se pueden utilizar para definir componentes nuevos.
- Proporcionar un modelo de JavaBeans para enviar eventos desde los controles de la interfaz de usuario del lado del cliente a la aplicación del servidor.
- Definir APIs para la validación de entrada, incluyendo soporte para la validación en el lado del cliente.

[26] Figura obtenida de http://en.wikipedia.org/wiki/File:20110510-jsf-logo.tiff

[27] Información obtenida de Wikipedia (WWW31).

- Especificar un modelo para la internacionalización y localización de la interfaz de usuario.
- Automatizar la generación de salidas apropiadas para el objetivo del cliente, teniendo en cuenta todos los datos de configuración disponibles del cliente, como versión del navegador.

### 2.5.5 Metodología RUP.


***Figura 10: Logotipo RUP[28]***

El Proceso Unificado de Rational conocido como RUP, es un proceso de desarrollo de software desarrollado por la empresa Rational Software, actualmente propiedad de IBM. Junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, diseño, implementación y documentación de sistemas orientados a objetos (WWW32).

Sus características principales son[29]:

- Forma disciplinada de asignar tareas y responsabilidades (quién hace qué, cuándo y cómo).
- Pretende implementar las mejores prácticas en Ingeniería de Software.
- Desarrollo iterativo e incremental.
- Administración de requisitos.

[28] Figura obtenida de http://metodologiadesistemasdeinformacion.wikispaces.com

[29] Información obtenida de Wikipedia (WWW32).

- Uso de arquitectura basada en componentes.
- Control de cambios.
- Modelado visual del software.
- Verificación de la calidad del software.
- Incluye artefactos y roles.

La estructura dinámica de RUP es la que permite que éste sea un proceso de desarrollo iterativo, y en esta parte se ven inmersas cuatro fases (WWW32):

1) Inicio (llamado también Incepción o Concepción). Definir el alcance del proyecto con los patrocinadores, identifica los riesgos asociados al proyecto, propone una visión muy general de la arquitectura de software y genera los planes de las fases y de iteraciones posteriores.

2) Elaboración. Se seleccionan y desarrollan los casos de uso, se realiza la especificación de los casos de uso seleccionados y el primer análisis del dominio del problema, se diseña la solución preliminar.

3) Desarrollo (también llamado Implementación, Construcción). Completar la funcionalidad del sistema, para ello se deben clarificar los requisitos pendientes, administrar los cambios de acuerdo a las evaluaciones realizados por los usuarios y se realizan las mejoras para el proyecto.

4) Cierre (llamada Fase de Transición). Asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a los usuarios y proveer el soporte técnico necesario.

# CAPÍTULO III

# 3 Patrones de Diseño

## 3.1 Introducción

***Figura 11: Desarrollador de software***[30]

El diseño es un modelo del sistema a desarrollar, elaborado con una serie de principios y técnicas, que permite su descripción con el suficiente detalle como para ser implementado. Sin embargo, aplicar principios y reglas no es suficiente. Lograr que un diseño sea flexible y reutilizable a la primera vez es casi imposible. Antes de terminar el diseño, este ha de ser probado y modificado varias veces.

> En el contexto del diseño, se puede observar que los buenos ingenieros conservan esquemas y estructuras de solución que son utilizadas numerosas veces en función de la condición del problema. Estos esquemas y estructuras son conceptos reusables, es decir, son soluciones que pueden ser reutilizadas ante problemas similares (A Pattern Language: Towns/Builder/Construction, 1977).

[30] Figura tomada de:
http://blog.prestario.com/2011/buscamos-desarrollador-experimentado-symfony/

Los patrones tratan de aprovechar la experiencia adquirida directa o indirectamente y resuelven problemas de diseño específicos, haciendo diseños más flexibles, elegantes y reutilizables.

## 3.2 Definición de Patrón

Diferentes autores han formulado varias definiciones para la descripción de patrones. Por ejemplo, en el libro Entendiendo y Usando Patrones en el Desarrollo de Software, sus autores los definen como: “Un patrón es la abstracción de una forma concreta que puede repetirse en contextos específicos” (Riehle & Züllighoven, 1996).

Richard Gabriel, autor de Patrones de Software: Cuentos de la Comunidad de Software, provee una definición más clara:

> Cada patrón es una regla de tres partes, la cual expresa una relación entre un cierto contexto, un conjunto de fuerzas que ocurren repetidamente en ese contexto y una cierta configuración de software que permite a estas fuerzas resolverse por sí mismas (Gabriel, 1996).

La definición más utilizada es la expuesta por Alexander:

> Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo ni siquiera dos veces de la misma forma (A Pattern Language: Towns/Builder/Construction, 1977).

Se puede concluir que, para cualquier ciencia o ingeniería, un patrón es un vocabulario común para expresar sus conceptos, y relacionarlos unos con otros; su objetivo principal dentro de la comunidad del software es ayudar a crear un lenguaje compartido, para comunicar puntos de vista y experiencia sobre problemas dentro de los procesos de desarrollo de software y sus soluciones.

### 3.3 Características de los Patrones

Para que una solución sea considerada un patrón debe pasar por pruebas y poseer ciertas características. La tabla 1, indica las características que los buenos patrones deben tener, de acuerdo al criterio de Jim Coplien:

| **Característica** | **Punto de vista** |
|---|---|
| Solucionar un problema | Los patrones capturan soluciones, no sólo principios o estrategias abstractas. |
| Ser un concepto probado | Los patrones capturan soluciones demostradas, no teorías o especulaciones. |
| La solución no es obvia | Los mejores patrones generan una solución a un problema de forma indirecta. |
| Describe participantes y relaciones entre ellos | Los patrones no sólo describen módulos sino estructuras del sistema y mecanismos más complejos. |
| El patrón tiene un componente humano significativo | El Software proporciona a los seres humanos confort o calidad de vida (estética y utilidad). |

***Tabla 1: Características de los patrones***[31]

[31] Información tomada del libro Patrones de Software (Coplien, 1996).

### 3.3.1 Abstracción de patrones.

Un patrón describe, con algún nivel de abstracción, una solución experta a un problema. Normalmente, un patrón está documentado en forma de una plantilla especializada y aunque es una práctica estándar, no significa que sea la única forma de hacerlo. Además, hay tantos formatos de plantillas como autores de patrones, esto permite la creatividad en la documentación de patrones (WWW33).

Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que describen soluciones para todo, desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

## 3.4 Tipos de Patrones

Existen diferentes ámbitos dentro de la Ingeniería del Software donde se pueden aplicar los patrones. A continuación se detallan algunos tipos de patrones:

### 3.4.1 Patrones de arquitectura.

Estos patrones indican formas de descomponer, conectar y relacionar sistemas, es decir, expresan una organización o esquema estructural fundamental para sistemas de software (WWW34).

Proporcionan un conjunto de subsistemas predefinidos, especifican sus responsabilidades, e incluyen una guía para organizar las relaciones entre ellos; por ejemplo: programación por capas, Modelo Vista Controlador, arquitectura orientada a servicios, entre otros.

### 3.4.2 Patrones de diseño.

"Proporcionan un esquema para refinar los subsistemas o componentes de un sistema software, o las relaciones entre ellos" (WWW34). Describen estructuras repetitivas de comunicación de componentes que resuelven un problema de diseño en un contexto particular.

### 3.4.3 Patrones de programación (Idioms patterns).

Son patrones de bajo nivel sobre un lenguaje de programación específico, los cuales describen la implementación de cuestiones concretas como aspectos de componentes o relaciones entre ellos, utilizando las facilidades del lenguaje de programación dado[32].

### 3.4.4 Patrones de análisis.

**"**Describen un conjunto de prácticas que aseguran la obtención de un buen modelo de un problema y su solución" (Coplien, 1996). Es decir, estos patrones definen reglas que permiten modelar un sistema satisfactoriamente y ayudan a elegir las clases adecuadas y la forma como estas deben interactuar. Son conocidos también como GRASP (Patrones Generales de Responsabilidad de Asignación de Software). Entre estos se pueden mencionar: bajo acoplamiento[33], alta cohesión[34].

[32] Información tomada del libro Desarrollo de Proyectos Informáticos con Tecnología Java (Óscar Belmonte, 2012).

[33] Es la idea de tener las clases lo menos ligadas entre sí, potenciando la reutilización y disminuyendo la dependencia entre clases (WWW35).

[34] Dice que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase (WWW35).

### 3.4.5 Patrones organizacionales.

Describen la estructura y prácticas de las organizaciones humanas, especialmente en las que producen, usan o administran software. El dominio de los patrones de diseño abarca el diseño de software, mientras que el dominio de los patrones organizacionales es la administración del software y sus procesos; por ejemplos el patrón gatekeeper[35].

### 3.4.6 Patrones de proceso.

Son una colección de técnicas generales, acciones, tareas o actividades para el desarrollo de software orientado a objetos. Se identifican tres tipos de patrones de proceso principales: task process[36], stage process[37] y phase process[38].

La diferencia entre las clases de patrones expuestas radica en los diferentes niveles de abstracción y detalle, y, del contexto particular en el cual se aplican o de la etapa en el proceso de desarrollo.

[35] **Gatekeeper,** define la forma de acceder a un sistema de almacenamiento con el objetivo de minimizar el área de ataque (WWW36).

[36] **Procesos de tarea,** define pasos detallados para realizar una tarea (Mejía).

[37] **Procesos de etapa,** pasos repetitivos para una simple etapa de desarrollo (Mejía).

[38] **Procesos de fase,** define las interacciones entre los patrones de etapa para el período de desarrollo (Mejía).

## 3.5 Qué es un Patrón de Diseño

Los patrones de diseño se pueden definir como esquemas predefinidos o conjunto de estrategias aplicables en diversas situaciones, de forma que el analista se asegura de que el diseño que está aplicando tiene ciertas cualidades que le proporcionan calidad.

El libro Design Patterns propone la siguiente definición para patrón de diseño: "Los patrones de diseño son descripciones de objetos y clases comunicándose, que son adaptadas para resolver un problema de diseño general en un contexto particular" (Erich Gamma, 1994). Por lo tanto, los patrones se centran en la micro-estructura de las aplicaciones, es decir, en sus clases y objetos.

Cada patrón de diseño trata de agrupar la experiencia en diseño de tal forma que los desarrolladores puedan utilizarlos con efectividad. Por eso se han documentado los más importantes patrones de diseño y presentado en catálogos.

## 3.6 Objetivos de los Patrones de Diseño

Los patrones de diseño pretenden (WWW34):

- Proporcionar catálogos de elementos reusables en el diseño de software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Análogamente, los patrones de diseño no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.

## 3.7 Estructura o plantillas de Patrones de Diseño

Para describir patrones de diseño, se utiliza un formato consistente que establezca un medio de comunicación homogéneo entre diseñadores. Varios autores han propuesto plantillas ligeramente distintas que especifican los mismos conceptos básicos.

Indistintamente de la plantilla que se elija, cada patrón es dividido en secciones. A continuación se señalan los apartados que integran una plantilla estándar[39]:

- **Nombre del patrón**: Esta sección consiste en un nombre estándar, y una referencia bibliográfica que indica la procedencia del patrón. El nombre es significativo y corto, fácil de recordar y asociar a la información subsecuente.
- **Contexto**: Describe el problema que el patrón soluciona. Este problema suele ser introducido en términos de un ejemplo concreto.
- **Solución**: Detalla una solución general al problema que el patrón soluciona. Esta descripción puede incluir diagramas y texto que identifique la estructura del patrón, sus participantes y sus colaboraciones.
- **Consecuencias**: Explica las implicaciones, buenas y malas, del uso de la solución.

[39] Información obtenida del libro Patrones de Diseño (Erich Gamma, 1994).

- **Implementación**: Describe las consideraciones importantes que se han de tener en cuenta cuando se codifica la solución. Puede contener algunas variaciones o simplificaciones de la solución.
- **Patrones relacionados**: Contiene una lista de los patrones que están relacionados con el patrón que se describe.

Un patrón nombra, abstrae e identifica los aspectos claves de la estructura de un diseño común, el cual es útil para crear un diseño reutilizable orientado a objetos. Identifica las clases participantes y sus instancias, los roles, colaboraciones, y la distribución de responsabilidades. Cada patrón de diseño se enfoca en un problema o situación específica.

### 3.8 Por qué son útiles los Patrones de Diseño

El principal objetivo de los patrones de diseño es capturar buenas prácticas que permitan mejorar la calidad del diseño de los sistemas, determinando objetos que soporten roles útiles en un contexto específico, encapsulando la complejidad y haciéndolo más flexible.

Los beneficios resultantes del uso de un patrón, pueden ser medidos en varios sentidos, por ejemplo:

- Contribuyen a reutilizar diseño, identificando aspectos claves de la estructura de un diseño que puede ser aplicado en diferentes situaciones. La reutilización del diseño es importante porque reduce los esfuerzos de desarrollo y mantenimiento, mejora la seguridad, eficiencia y consistencia de diseños.
- Mejoran la flexibilidad, modularidad y extensibilidad; factores internos e íntimamente relacionados con la calidad percibida por el usuario.

- Incrementan el vocabulario de diseño de los desarrolladores, ayudando a diseñar desde un mayor nivel de abstracción.

## 3.9 Clasificación de los Patrones de Diseño

Diferentes autores han propuesto varias categorías para clasificar los patrones de diseño. Sin embargo, la clasificación mayoritariamente aceptada por la comunidad de desarrolladores es la propuesta por GoF[40]. Esta clasificación está apoyada en uno de dos criterios.

El primero se basa en el propósito del patrón y refleja lo que este hace. El segundo criterio se basa en el alcance del patrón, es decir, en qué se aplica principalmente, en las clases o en los objetos.

Según el propósito, los patrones de diseño se clasifican en tres categorías, descritas en la tabla 2.

[40] **Gang of Four (GoF, pandilla de los cuatro),** formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides autores del libro Design Patterns (WWW38)

| Categorías | Descripción |
|---|---|
| Patrones Creacionales | Se enfocan en la creación o instanciación de objetos.<br>Abstraen el proceso de instanciación.<br>Ocultan detalles de creación o inicialización. |
| Patrones Estructurales | Manejan la composición de clases y objetos.<br>Describen la forma de combinarlos para proporcionar funcionalidad y obtener grandes estructuras.<br>Los objetos adicionados pueden ser incluso objetos simples u objetos compuestos. |
| Patrones de Comportamiento | Indican formas en que clases y objetos interactúan y asignan responsabilidades.<br>Definen comunicación e iteración entre objetos del sistema.<br>Pretenden disminuir el acoplamiento entre objetos. |

***Tabla 2: Descripción de las categorías de los patrones según el propósito***[41]

En cada categoría existen dos estrategias, una empleando herencia, llamada *Estrategia de Clase,* y otra empleando asociaciones de objetos, *Estrategia de Objeto*, por lo cual, de acuerdo al segundo criterio mencionado los patrones se clasifican en dos ámbitos: clases y objetos, detallados en la tabla 3.

[41] Información obtenida del libro Design Patterns (Erich Gamma, 1994).

| Categorías | Descripción |
|---|---|
| Creacional de Clase | Manejan relaciones entre clases y subclases.<br>Las relaciones son establecidas mediante herencia (forma estática).<br>Son resueltas en tiempo de compilación como un mecanismo para lograr la instanciación de la clase. |
| Creacional de Objeto | Manejan relaciones entre objetos.<br>Son más escalables y dinámicos que los patrones creacionales de clase.<br>Objetos cambian en tiempo de corrida. |
| Estructural de Clase | Utilizan la herencia para componer otras clases y proporcionar interfaces más útiles combinando la funcionalidad de múltiples clases. |
| Estructural de Objeto | Describen formas de ensamblar objetos.<br>Crean objetos complejos agregando objetos individuales.<br>Su composición puede cambiar en tiempo de ejecución, proporcionando mayor flexibilidad que los patrones estructurales de clase. |
| Comportamiento de Clase | Utilizan la herencia para distribuir el comportamiento entre clases, es decir, para describir algoritmos y flujos de control. |
| Comportamiento de Objeto | Analizan la comunicación entre objetos interconectados.<br>Describen cómo un grupo de objetos coopera para realizar una tarea que solos no pueden desempeñar. |

***Tabla 3: Descripción de las categorías de los patrones según el alcance***[42]

[42] Información obtenida del libro Design Patterns (Erich Gamma, 1994).

En la tabla 4, se puede observar la clasificación de estos patrones de acuerdo a las categorías expuestas.

| | | PROPÓSITO | | |
|---|---|---|---|---|
| | | **CREACIÓN** | **ESTRUCTURA** | **COMPORTAMIENTO** |
| **ALCANCE** | **CLASE** | Factory Method | Adapter (Clase) | Interpreter<br>Template Method |
| | **OBJETO** | Abstract Factory<br>Builder<br>Prototype<br>Singleton | Adapter (Objeto)<br>Bridge<br>Composite<br>Decorator<br>Facade<br>Flywheight<br>Proxy | Chain of Responsability<br>Command<br>Memento<br>Observer<br>State<br>Iterator<br>Mediator<br>Strategy<br>Visitor |

***Tabla 4: Clasificación de patrones de diseño GoF***[43]

## 3.10 Problemas de diseño que los patrones ayudan a resolver

A continuación se describen algunos problemas que los diseñadores enfrentan, y la forma en que los patrones de diseño ayudan a resolverlos.

### 3.10.1 Encontrar los objetos apropiados.

Una de las partes más difíciles del diseño orientado a objetos es descomponer un sistema en objetos, ya que se debe tomar en cuenta la encapsulación, granularidad, dependencia, flexibilidad, desempeño y reutilización.

[43] Información obtenida del libro Design Patterns (Erich Gamma, 1994).

Los patrones de diseño ayudan a identificar las abstracciones menos obvias y los objetos que las representan, por ejemplo, objetos que representan un proceso o algoritmo y no tienen una representación física en un sistema real, pero son parte crucial para un diseño flexible.

### 3.10.2 Determinar la granularidad de un objeto.

Otro problema al que se enfrentan los diseñadores es establecer qué debe ser un objeto y qué no. Existen patrones de diseño que describen la forma de representar subsistemas como objetos, y la manera con que se puede manejar un gran número de ellos; patrones que describen cómo descomponer un objeto en objetos más pequeños y patrones que producen objetos responsables de implementar peticiones en otros objetos o grupos de ellos.

### 3.10.3 Especificar la interfaz de un objeto.

La definición de la interfaz de un objeto es otra situación en la que los patrones son de mucha utilidad porque ayudan a definir interfaces, identificando elementos clave y los datos que son enviados a través de dichas interfaces. Existen patrones que describen la forma de encapsular y guardar el estado interno de un objeto para que pueda ser restaurado a ese estado posteriormente. Algunos patrones también especifican relaciones entre interfaces.

### 3.10.4 Especificar la implementación de un objeto.

La implementación de un objeto está definida por su clase, y los patrones, ayudan a especificar la implementación de un objeto. Existen patrones de diseño en los cuales los objetos deben compartir un tipo común, pero usualmente no comparten una misma implementación. Otros patrones abstraen el proceso de creación de objetos y ofrecen diferentes formas de asociar una interfaz con su implementación, de forma transparente en el momento de la creación o instanciación del objeto. Los patrones de creación ayudan a que el sistema escrito esté desarrollado en términos de interfaces y no en clases concretas.

### 3.10.5 Relación de estructuras dinámicas y estáticas.

El código de la estructura estática (compile time) consiste en clases relacionadas mediante herencia, y la estructura dinámica (run time) consiste en redes de comunicación de objetos que cambian rápidamente. La estructura dinámica debe ser impuesta por el diseñador, no por lenguaje, determinando de la forma más adecuada las relaciones entre objetos y sus tipos, ya que de esto depende qué tan buena o mala sea esta estructura en un programa.

Algunos patrones de diseño capturan explícitamente la diferencia entre las estructuras en tiempo de compilación y en tiempo de ejecución, siendo útiles para construir complejas estructuras dinámicas.

### 3.10.6 Diseñar anticipándose a los cambios.

La clave para maximizar la reutilización está en anticiparse a los nuevos requerimientos y a cambios en los ya existentes. Un diseño que no tome en cuenta la evolución del sistema, aumenta el riesgo de que exista la necesidad de realizar modificaciones futuras al diseño. Los patrones ayudan a evitar el rediseño, asegurándose que un sistema pueda cambiar o evolucionar en formas específicas.

## 3.11 Directrices para el uso de un Patrón de Diseño

No existe una guía para aplicar los patrones de diseño, sólo la experiencia indicará la forma correcta de emplearlos, sin embargo, se presentan ciertas claves que se pueden tomar en cuenta para utilizar un patrón[44]:

- **Leer el documento que describe el patrón.** Para tener una idea de lo que indica el patrón, poniendo especial atención a las secciones contexto y consecuencias.

- **Observar la estructura, los elementos que participan en el patrón y las colaboraciones entre ellos.** A fin de comprender de manera correcta las clases y objetos que integran el patrón y la forma en la que se relacionan.

- **Definir las clases, declarar sus interfaces, establecer las relaciones de herencia y definir instancias para representar datos y referencias a objetos.** Identificar las clases de la aplicación que el patrón afectará y modificará adecuadamente.

[44] Información tomada del libro Design Patterns J2EE (William Crawford, 2003).

- **Definir nombres específicos para las operaciones en la aplicación.** Los nombres dependen generalmente de la aplicación. Para nombrar operaciones se debe ser consistente en la convención.

- **Implementar las operaciones para definir las responsabilidades y colaboraciones en el patrón.** La sección de implementación ofrece tips o técnicas que sirven de guía en la implementación.

### 3.12 Antipatrón de Diseño

"Un antipatrón de diseño es un patrón de diseño que invariablemente conduce a una mala solución para un problema" (WWW39).

Los antipatrones son considerados como parte importante de una buena práctica de programación. Un buen programador, intentará evitar el uso de antipatrones, lo cual requiere su reconocimiento e identificación temprana, dentro del ciclo de vida del software.

Pueden ser clasificados en: antipatrones de gestión de proyectos, de diseño de software, de diseño orientado a objetos, antipatrones de programación, entre otros. La tabla 5 muestra ejemplos de ellos:

| Tipo de antipatrón | Antipatrón | Descripción |
|---|---|---|
| Antipatrones de gestión de proyectos. | Humo y espejos (smoke and mirrors) | Mostrar cómo será una funcionalidad antes de su implementación. |
| | Software inflado (software bloat) | Permitir que las versiones de un sistema exijan cada vez más recursos. |
| Antipatrones de diseño de software. | Clase Gorda | Dotar a una clase con demasiados atributos y/o métodos, haciéndola responsable de la mayoría de la lógica de negocio. |
| | Re-dependencia (re-coupling) | Introducir dependencias innecesarias entre objetos. |
| Antipatrones de diseño orientado a objetos. | Acoplamiento secuencial (sequential coupling) | Construir una clase que necesita que sus métodos sean invocados en un orden determinado. |
| | Modelo de dominio anémico (anemic domain model) | Usar un modelo del dominio sin ninguna lógica de negocio. |
| Antipatrones de programación. | Código duro (hard code) | Hacer supuestos sobre el entorno del sistema en demasiados lugares de la implementación. |
| | Manejo de excepciones (exception handling) | Emplear el mecanismo de manejo de excepciones del lenguaje para implementar la lógica general del programa. |

***Tabla 5: Antipatrones de diseño[45]***

[45] Información tomada de Wikipedia (WWW39).

## 3.13 Patrones de Diseño JEE

Frecuentemente se construyen aplicaciones web en las que cada página JSP gestiona servicios de seguridad, de recuperación de contenidos, y la navegación. Esto crea un modelo con un alto coste de mantenimiento, debido a la existencia de código duplicado, generalmente, por el uso de la técnica de copiar y pegar.

Se puede mejorar significativamente la calidad de estas aplicaciones centralizando y encapsulando algunos mecanismos, haciendo la aplicación mucho más mantenible, sencilla, y limpia. Para conseguir estos objetivos no hay nada mejor que la experiencia condensada de muchos años de desarrollo y diseño: los patrones de diseño JEE.

Con la aparición de JEE versión 6, un nuevo catálogo de patrones de diseño fue difundido, proporcionando los beneficios de esta plataforma, y, soluciones para los problemas típicamente encontrados por arquitectos y diseñadores en el proceso de desarrollo de aplicaciones.

### 3.13.1 Qué son los patrones de diseño JEE.

"Los patrones de diseño JEE contienen las mejores soluciones para ayudar a los desarrolladores a diseñar y construir aplicaciones en la plataforma JEE" (Bien, 2009).

Aunque estos patrones son representados desde un nivel lógico de abstracción, todos contienen en sus descripciones originales, varias estrategias que indican detalles para su implementación.

### 3.13.2 Diseño de aplicaciones web con patrones de diseño JEE.

La plataforma JEE ofrece distintas posibilidades de diseño. Sin embargo, la utilización del patrón arquitectónico MVC y los Patrones de Diseño JEE dan a las aplicaciones escalabilidad, facilitando futuras ampliaciones, migraciones y cambios en general.

El diseño de aplicaciones Web basado en patrones JEE, se organiza intrínsecamente alrededor de la utilización de varios elementos: un controlador frontal[46], despachadores[47], vistas compuestas, vistas estáticas (JSPs) y los ayudantes[48] de las vistas dinámicas formadas con JavaBeans[49]. Esto puede relejarse en la figura 12, que ejemplifica un diseño basado en estos mecanismos.

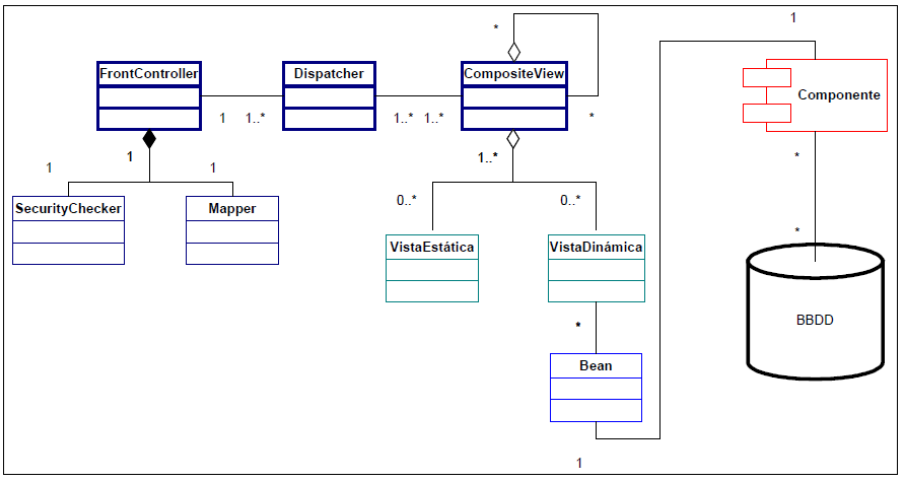


***Figura 12: Diseño basado en patrones JEE[50]***

[46] **Front Controller,** consiste en utilizar un único punto de entrada a cada aplicación (WWW40).

[47] **Dispatcher,** parte de un programa encargada de lanzar un proceso en el servidor de un entorno cliente/servidor (WWW41).

[48] **Helper,** facilita la reutilización del código escrito (Universidad de Alicante, 2011)

[49] **JavaBeans,** componentes de software reutilizables que se puedan manipular visualmente en una herramienta de construcción (Wikipedia, 2012).

[50] Figura tomada del manual Diseño de Aplicaciones Web (Díaz, 2003)

### 3.13.3 Importancia del uso de patrones de diseño en aplicaciones web.

Los patrones de diseño son importantes para garantizar de la mejor forma que una aplicación web sea:

- **Extensible**. Se determina en la forma en que la aplicación puede crecer para incorporar nuevos requisitos de los usuarios, no sólo desde el punto de vista operativo sino también técnico; no debe ser difícil para los programadores ampliar las funcionalidades y no deben repetir código fuente.

- **Escalable**. Esto se determina en la forma en que la aplicación pueda soportar mayor cantidad de usuarios y peticiones en un ambiente exigente. La escalabilidad está relacionada directamente con el rendimiento.

- **Fiable.** Se comprueba no sólo por la calidad del software, sino porque el hardware y la red se comporten de la forma esperada.

- **Puntual en tiempos del proyecto.** La meta en cualquier proyecto de software es realizar una entrega en el tiempo proyectado, es lo que más le interesa al usuario final.

### 3.13.4 Catálogo de patrones de diseño JEE 6.

La mayoría de los proyectos Java EE 6 continúan utilizando patrones J2EE pero esta plataforma fue intrusiva, la separación de la infraestructura técnica y la lógica de negocio, necesitaba la introducción obligatoria de patrones, que eran en su mayoría soluciones para las deficiencias de J2EE.

La funcionalidad de esta nueva versión de la plataforma Empresarial de Java, demanda un replanteamiento de los patrones de diseño utilizados en J2EE, así como la eliminación de código superfluo requerido para la implementación de aplicaciones en dicha plataforma.

Tomando en cuenta el punto anterior y siguiendo la lógica de los patrones de diseño del GoF, aparecen una serie de patrones específicos del mundo Web que se distribuyen en tres capas: Presentación, Negocio e Integración.

#### *3.13.4.1 Patrones de la capa de presentación*.

La capa de presentación incluye Servlets, JSPs y otros elementos que generan interfaz de usuario del lado del servidor.

Los patrones de diseño que se exponen a continuación, forman parte del catálogo de J2EE pero pueden ser utilizados en JEE, sin embargo no hay que olvidar los beneficios que esta plataforma en su versión 6 ofrece, por lo que la utilización de los patrones en esta capa debe estar adecuadamente justificada.

3.13.4.1.1 *Intercepting Filter.*

Este patrón maneja varios tipos de peticiones que requieren un procesamiento determinado; es aplicado especialmente en procesos de validación de sesión y debe satisfacer las siguientes necesidades:

- Se requiere el pre-procesamiento y post-procesamiento de peticiones o respuestas de un cliente Web.
- Procesamiento común, como el chequeo del esquema de codificación de datos o la información completa de login de cada petición.
- Se desea la centralización de la lógica común.
- Debería facilitar la adición o eliminación de servicios sin afectar los componentes existentes, para que se puedan utilizar en diversas combinaciones, como:
  - Logging y autentificación.
  - Depuración y transformación de la salida para un cliente específico.
  - Descomprimir y convertir el esquema de codificación de la entrada.

Para esto se crean filtros conectables que procesan servicios comunes de una forma estándar, sin requerir cambios en el código principal del procesamiento de la petición. “Los filtros interceptan las peticiones entrantes y las respuestas salientes, permitiendo un pre y post-procesamiento; pueden ser añadidos y eliminados a discreción sin modificar el código existente” (Sun Microsystems, Inc, 2002).

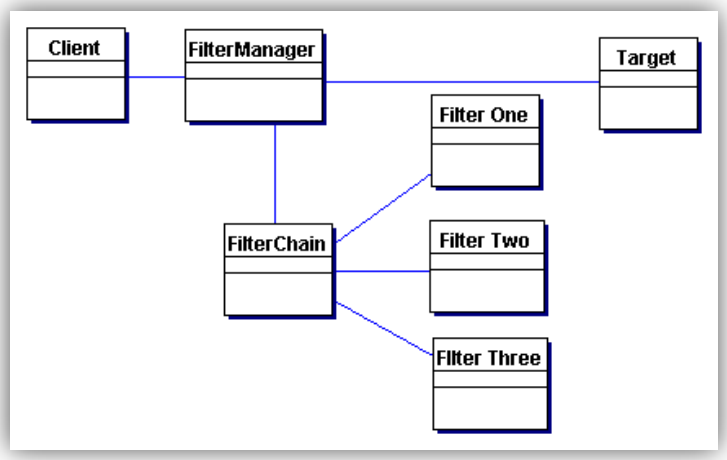


***Figura 13: Diagrama de clases del patrón Intercepting Filter***[51]

### *Características*

- Los filtros proporcionan un lugar centralizado para controlar el procesamiento a través de múltiples peticiones.
- Los filtros son añadidos o eliminados del código existente de forma transparente y debido a su interfaz estándar, funcionan en cualquier combinación y son reutilizables para varias presentaciones.
- Se pueden combinar numerosos servicios en varias permutaciones sin tener que recompilar el código fuente.
- Compartir información entre filtros puede ser ineficiente, ya que por definición, todo filtro tiene acoplamiento ligero.

### *Estrategias de implementación*

Para la implementación del patrón Intercepting Filter, se pueden aplicar las estrategias señaladas en la tabla 6.

[51] Figura obtenida del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

| Estrategia | Características |
|---|---|
| Custom Filter | Filtro implementado con una estrategia personalizada definida por el desarrollador.<br>Menos flexible y menos potente que la estrategia Standard Filter.<br>No provee una envoltura para los objetos request y response de forma portable.<br>El objeto request no puede ser modificado.<br>Se debe introducir un mecanismo de buffer si los filtros son utilizados para controlar el stream de salida.<br>El desarrollador podría utilizar el patrón Decorator[52] para envolver los filtros alrededor de la lógica principal del procesamiento de la petición. |
| Standard Filter | Filtros controlados de forma declarativa utilizando un descriptor de despliegue[53].<br>Filtros construidos sobre interfaces, y añadidos o eliminados declarativamente modificando el descriptor de despliegue de una aplicación Web. |
| Base Filter | Un filtro base sirve como una superclase común para todos los filtros.<br>Características comunes pueden ser encapsuladas en el filtro base y compartidas entre todos los filtros. |
| Template Filter | Usa un filtro base del que descienden otros filtros.<br>El filtro base se utiliza para dictar los pasos generales que cada filtro debe completar.<br>Deja detalles específicos de cómo completar los pasos en la subclase de cada filtro.<br>Impone una estructura para el procesamiento de los filtros.<br>Puede ser combinada con las estrategias anteriores. |

***Tabla 6: Características de las estrategias de implementación del patrón Intercepting Filter[54]***

[52] **Decorator (envoltorio),** añade dinámicamente funcionalidad a una clase.

[53] **Descriptor de despliegue,** componente de aplicaciones J2EE que describe cómo se debe desplegar una aplicación web. Este término también se usa como referencia a un fichero de configuración para un artefacto que es desplegado en algún contenedor/motor (Wikipedia, 2012).

[54] Información tomada del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

La tabla 7 muestra los patrones que se relacionan con el patrón en estudio.

| Patrones relacionados | Descripción |
|---|---|
| Front Controller | Resuelve problemas similares, pero es más adecuado para el manejo del procesamiento central. |
| Decorator | Proporciona envolturas dinámicamente conectables. |
| Template Method | Utilizado para implementar la estrategia Template Filter. |

***Tabla 7: Patrones relacionados con el patrón Intercepting Filter[55]***

3.13.4.1.2 *Front Controller.*

El patrón Front Controller acepta todas las peticiones de un cliente y las direcciona a los manejadores apropiados. Debe satisfacer las siguientes exigencias:

- El sistema necesita un punto de acceso centralizado que permita la administración de las peticiones de la capa de presentación, soporte la integración de los servicios del sistema, la recuperación de contenidos, control de vistas, y navegación.
- Se desea evitar la duplicación de la lógica de control.
- Es necesario aplicar la lógica común a varias solicitudes.
- Se requiere separar la vista de la lógica de procesamiento del sistema.
- Se necesita centralizar los puntos de acceso controlados en el sistema.

[55] Información obtenida del sitio web del libro Core J2EE Patterns (WWW47).

El patrón Front Controller centraliza la lógica de control impidiendo su duplicación, y, administra la invocación de los servicios de seguridad como la autentificación, controla la elección de una vista apropiada y el manejo de errores (Sun Microsystems, Inc, 2002).

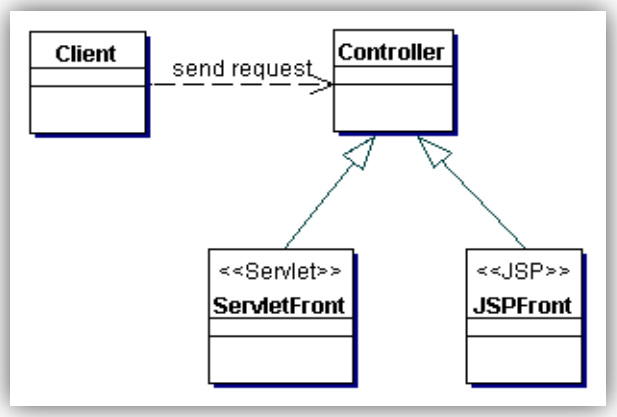


***Figura 14: Diagrama de clases del patrón Front Controller***[56]

### *Características*

- Un controlador administra el procesamiento de la lógica de negocio y el manejo de las peticiones.
- Provee un punto de obstrucción para los intentos de acceso ilícitos en la aplicación Web.
- Un controlador promueve el particionamiento transparente de la aplicación y fomenta la reutilización, ya que el código que es común entre los componentes se desplaza dentro de un controlador o es gestionado por este.

[56] Figura tomada del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

### *Estrategias de implementación*

Existen varias estrategias para implementar un controlador, la tabla 8 describe las principales:

| Estrategia | Características |
|---|---|
| Servlet Front | Sugiere la implementación del controlador como un servlet.<br>El controlador administra aspectos del manejo de la petición que están relacionados con el procesamiento del negocio y el control de flujo. |
| JSP Front | Propone la implementación del controlador como una página JSP.<br>No es adecuada porque:<br>- El controlador maneja el procesamiento que no está relacionado con el formateo de la salida.<br>- Requiere que el desarrollador trabaje con una página de etiquetas para modificar la lógica del control de peticiones. |
| Base Front | Es utilizada conjuntamente con la estrategia Servlet Front.<br>Propone la implementación de una clase base controladora, cuya implementación puede extenderse a otros controladores.<br>Los cambios realizados en una subclase afectan a las demás. |

***Tabla 8: Características de las estrategias de implementación del patrón Front Controller[57]***

Los patrones relacionados con el Front Controller son descritos en la tabla 9.

[57] Información obtenida del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

| Patrones relacionados | Descripción |
|---|---|
| View Helper | En conjunto, describe la creación de la lógica de negocio de la vista y suministra un punto central de control y envío.<br>El flujo lógico se construye dentro del controlador y el código de manejo de datos se transporta hacia los helpers. |
| Intercepting Filter | Igualmente, describe formas de centralizar el control de ciertos aspectos del procesamiento de peticiones. |
| Dispatcher View y Service to Worker | Son la combinación del View Helper con un dispatcher y un patrón Front Controller. |
| Service to Worker | Es una arquitectura centrada en el controlador resaltando el uso del Front Controller. |
| Dispatcher View | Es una arquitectura centrada en la vista. |

***Tabla 9: Patrones relacionados con el patrón Front Controller[58]***

3.13.4.1.3 *View Helper.*

El patrón View Helper encapsula la lógica de acceso a bases de datos. Debe solventar los siguientes requerimientos:

- Se desea utilizar plantillas basadas en vistas, como JSPs.
- Es necesario evitar la incorporación de la programación en la vista.
- Se requiere separar la lógica de programación de la vista, para facilitar la división del trabajo entre desarrolladores de software y diseñadores de páginas web.

[58] Información obtenida del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

Para solventar estas exigencias se utilizan vistas que encapsulen el código de formateo y helpers para encapsular la lógica del procesamiento de la vista. Una vista delega el procesamiento a sus clases helper, implementadas como JavaBeans o etiquetas personalizadas. Los helpers sirven como adaptadores entre la vista y el modelo de negocio, y realizan el procesamiento relacionado con la lógica de formateo, como la generación de una tabla HTML (Sun Microsystems, Inc, 2002).

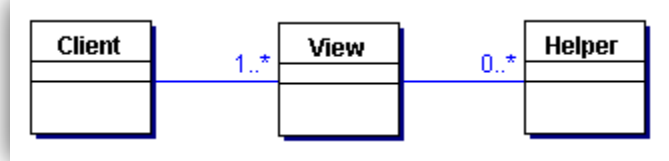


***Figura 15: Diagrama de clases del patrón View Helper***[59]

### *Características*

- El uso de helpers separa la vista del procesamiento de negocio en una aplicación. Proporcionan un lugar externo para que la vista encapsule la lógica de negocio, que es construida fuera de las JSPs y dentro de JavaBeans o etiquetas personalizadas, y, es reutilizada para disminuir la duplicación y facilitar el mantenimiento.
- Reduce las dependencias que los individuos que cumplen diferentes funciones pueden tener en los mismos recursos.

[59] Figura tomada del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

### *Estrategias de implementación*

Existen varias estrategias para implementar el View Helper, y a continuación son descritas en la tabla 10:

| **Estrategia** | **Características** |
|---|---|
| JSP View | Sugiere utilizar una página JSP como el componente vista.<br>Las vistas son el dominio de los diseñadores Web, que prefieren un lenguaje de marcas al código Java. |
| Servlet View | Utiliza un servlet como la vista.<br>Embebe etiquetas de marcas dentro del código Java.<br>La fusión de etiquetas y código, hace que la plantilla de la vista sea más difícil de actualizar y modificar. |
| JavaBean Helper | El helper es implementado como un JavaBean.<br>La lógica de negocio está construida fuera de la vista y dentro del componente helper.<br>El encapsulamiento de la lógica de negocio en un JavaBean, favorece la recuperación de contenido y, adapta y almacena el modelo para usarlo en la vista. |
| Custom Tag Helper | El helper se implementa como una etiqueta personalizada.<br>La lógica de negocio se construye fuera de la vista y dentro del componente helper.<br>La lógica de negocio se encapsula en un componente de etiqueta personalizada. |

***Tabla 10: Características de las estrategias de implementación del patrón View Helper[60]***

[60] Información obtenida del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

La tabla 11 muestra los patrones relacionados con el patrón View Helper.

| Patrones relacionados | Descripción |
|---|---|
| Business Delegate | Oculta al cliente los detalles subyacentes de la búsqueda y acceso a los servicios de negocio, y, proporciona un caché intermedio para reducir el tráfico de la red. |
| Dispatcher View o Service to Worker | Se usan cuando se necesita un control centralizado para el manejo de problemas como la seguridad, administración de flujo de trabajo, y, recuperación de contenidos y navegación. |
| View Helper | Se asocia con el Front Controller para crear los patrones anteriores. |

***Tabla 11: Patrones relacionados al patrón View Helper***[61]

3.13.4.1.4 *Composite View.*

El patrón Composite View debe satisfacer las siguientes exigencias:

- Se requiere subvistas comunes, como encabezados y pies de página, en múltiples vistas, que pueden aparecer en diferentes lugares dentro de cada diseño de página.
- Existe contenido en las subvistas que cambia frecuentemente o puede estar sujeto a ciertos controles de acceso.
- Se desea evitar la incrustación directa y la duplicación de subvistas en varias vistas.

[61] Información obtenida del sitio web del libro Core J2EE Patterns (WWW47).

Para solventar estas necesidades, se utilizan Vistas Compuestas conformadas por varias subvistas. Cada componente de la plantilla se puede incluir dinámicamente y el diseño de la página es administrada independientemente del contenido.

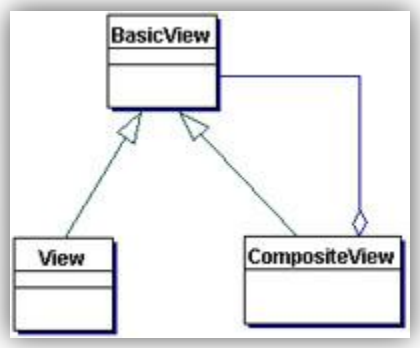


***Figura 16: Diagrama de clases del patrón Composite View***[62]

### *Características*

- Promueve un diseño modular. Es posible reutilizar fragmentos de una plantilla, en varias vistas y re-decorar estos fragmentos con información diferente.
- Una implementación sofisticada puede incluir fragmentos de una plantilla de vista basándose en decisiones en tiempo de ejecución, como roles de usuario o políticas de seguridad.
- Es más eficiente manejar cambios en partes de una plantilla cuando esta no es codificada directamente en las marcas de la vista. Las modificaciones en el diseño de una página también se gestionan fácilmente porque los cambios están centralizados.

[62] Figura tomada del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

- El administrador de la vista maneja la introducción de fragmentos de la plantilla en la vista compuesta.
- Puede ser parte de un motor de ejecución de páginas JSP, o, estar encapsulado en un helper JavaBean o en una etiqueta personalizada para proveer una funcionalidad más robusta.

La tabla 12 indica los patrones que están relacionados con el Composite View.

| Patrones relacionados | Descripción |
|---|---|
| View Helper | El patrón Composite View podría ser utilizado como su vista. |
| Composite[63] (GoF) | El patrón Composite View se base en este patrón. |

***Tabla 12: Patrones relacionados con el patrón View Helper[64]***

### *Estrategias de implementación*

Para la implementación del patrón Composite View, se pueden utilizar las estrategias JSP View y Servlet View del patrón View Helper, además, las estrategias descritas en la tabla 13.

[63] **Composite,** permite tratar objetos compuestos como si fueran un objeto simple.

[64] Información obtenida del sitio web del libro Core J2EE Patterns (WWW47).

| Estrategia | Características |
|---|---|
| JavaBean View Management | Para la administración de la vista se utilizan componentes JavaBeans.<br>Los JavaBeans implementan la lógica para controlar la distribución y composición de la vista.<br>Las decisiones sobre el diseño de la página se basan en roles de usuario o políticas de seguridad. |
| Standard Tag View Management | El control de la vista se implementa utilizando etiquetas JSP estándar.<br>No proporciona potencia y flexibilidad porque el diseño de las páginas individuales permanece embebido dentro de esas páginas.<br>Cuando se crea una vista compuesta utilizando etiquetas estándar, se puede incluir contenido estático y dinámico. |
| Custom Tag View Management | El control de la vista es implementado mediante etiquetas personalizadas.<br>La lógica creada dentro de la etiqueta controla el diseño y la composición de la vista.<br>Las acciones personalizadas pueden basarse en la distribución y composición de la página, o, en roles del usuario y políticas de seguridad. |
| Transformer View Management | La administración de la vista se implementa utilizando un Transformador XSL.<br>Se complementa con el uso de etiquetas personalizadas para implementar y delegar en los componentes adecuados. |

***Tabla 13: Características de las estrategias de implementación del patrón Composite View***[65]

[65] Información obtenida del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

3.13.4.1.5 *Service to Worker.*

Este patrón es similar al patrón arquitectónico MVC, utiliza los patrones Front Controller para el controlador y View Helper para la vista con un componente dispatcher.

Con el uso de este patrón se pretende ejecutar la administración central de la petición e invocar la lógica de negocio antes de que el control pase a la vista. Debe solventar los siguientes requerimientos:

- Se requiere ejecutar la lógica de negocio específica para atender una petición, con el fin recuperar el contenido que se utilizará en la generación de una respuesta dinámica.
- Es necesario observar las opciones que pueden depender de las respuestas de las invocaciones del servicio de negocio.
- Es posible la utilización de un framework o librería en la aplicación.

***Características***

- Combina un controlador y un dispatcher con vistas y helpers con el fin de administrar las peticiones de los clientes y generar una presentación dinámica como respuesta.
- Los controladores delegan la recuperación del contenido a los helpers, que gestionan el modelo intermedio para la vista.
- Un dispatcher es el responsable de la administración de la vista y la navegación, puede encapsularse dentro de un controlador o es un componente independiente que trabaja en coordinación con el controlador.

- Proporciona un lugar central para manejar los servicios del sistema y la lógica de negocio entre varias peticiones. El patrón también promueve el particionamiento de la aplicación y la reutilización.
- El uso de helpers genera separación entre la vista y el procesamiento del negocio en una aplicación.
- Con la disgregación de la vista de la lógica de negocio, se reducen las dependencias de los recursos entre individuos que cumplen diferentes roles.

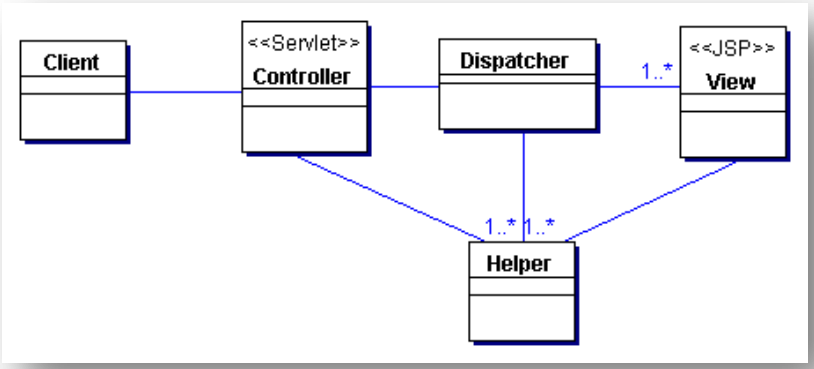


***Figura 17: Diagrama de clases del patrón Service to Worker***[66]

Mientras más responsabilidades encapsule el dispatcher, más se acercará al patrón Service to Worker ideal, y si tiene un rol más limitado, se aproximará a un perfecto Dispatcher View.

[66] Figura tomada del artículo Catálogo de Patrones de Diseño (Torrijos).

***Estrategias de implementación***

Las estrategias Servlet Front y JSP Front del patrón Front Controller pueden ser aplicadas para implementar el patrón Service to Worker, al igual que las estrategias JSP View, Servlet View, JavaBean Helper y Custom Tag Helper del patrón View Helper.

Las relaciones de este patrón con los otros patrones del catálogo expuesto, se indican en la tabla 14:

| Patrones relacionados | Descripción |
|---|---|
| View Helper con un dispatcher, coordinados con el Front Controller | Esta combinación crea el patrón Service to Worker. |
| Dispatcher View | Es igual respecto a los componentes implicados, pero diferente en la distribución de tareas entre dichos componentes. |

***Tabla 14: Patrones relacionados con el patrón Service to Worker***[67]

3.15.4.1.6 *Dispatcher View.*

Este patrón es similar al Service to Worker, pero el controlador no realiza acciones sobre el helper.

Su objetivo es crear una vista para manejar una petición y generar una respuesta, mientras se administran cantidades limitadas de procesamiento de negocio. Tiene las siguientes características:

[67] Información obtenida del sitio web del libro Core J2EE Patterns (WWW47).

- Posee vistas estáticas.
- Tiene vistas generadas a partir de un modelo de presentación existente.
- Existen vistas que son independientes de cualquier respuesta del servicio de negocio.
- Existe un limitado procesamiento del negocio.
- El controlador gestiona el procesamiento de la lógica de negocio y el manejo de peticiones. Maneja la autentificación y la autorización, y, delega en un dispatcher la gestión de la vista y la navegación.
- El dispatcher puede estar encapsulado en un controlador o ser un elemento independiente que trabaja conjuntamente con él.
- Un helper es el responsable de ayudar a la vista o al controlador a completar su procesamiento.
- El uso de helpers genera separación entre la vista y el procesamiento del negocio en una aplicación.
- Con la separación existente entre la vista y la lógica de negocio, se reducen las dependencias de los recursos entre individuos que cumplen diferentes roles.

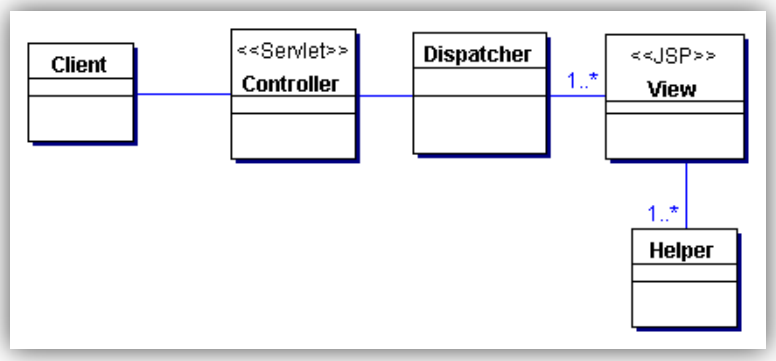


***Figura 18: Diagrama de clases del patrón Dispatcher View***[68]

[68] Figura tomada del artículo Catálogo de Patrones de Diseño (Torrijos).

### *Estrategias de implementación*

Las estrategias Servlet Front y JSP Front del patrón Front Controller pueden ser utilizadas para implementar el patrón Dispatcher View, al igual que las estrategias JSP View, Servlet View, JavaBean Helper y Custom Tag Helper del patrón View Helper.

Se puede utilizar también la estrategia Dispatcher in View en la cual, si el controlador fuera eliminado debido a su rol limitado, el dispatcher puede ser desplazado a una vista. Este diseño es útil en casos donde una vista mapee a una petición específica, pero el uso de una vista secundaria sea poco frecuente.

La tabla 15 muestra los patrones que se relacionan con el patrón Dispatcher View.

| Patrones relacionados | Descripción |
|---|---|
| View Helper con un dispatcher, coordinados con el Front Controller | Esta combinación crea el patrón Dispatcher View. |
| Service to Worker | Difiere con este patrón ya que posterga la recuperación del contenido al momento en que se procesa la vista. Además, el dispatcher asume un rol más limitado en el control de la vista, debido a que su elección ya está incluida en la petición. |

***Tabla 15: Patrones relacionados con el patrón Dispatcher View***[69]

[69] Información obtenida del Catálogo BluePrins (Sun Microsystems, Inc, 2002).

La tabla 16, es una matriz con las consecuencias que puede generar el uso de los patrones de diseño de la capa de presentación.

| | CONSECUENCIAS | | | | | |
|---|---|---|---|---|---|---|
| **PATRONES** | Centraliza el control | Mejora el mantenimiento | Mejora la reusabilidad | Mejora el particionamiento | Mejora la flexibilidad | Mejora la separación de roles |
| Intercepting Filter | X | — | X | — | X | — |
| Front Controller | X | X | X | — | — | — |
| View Helper | — | X | X | X | — | X |
| Composite View | — | X | X | X | X | — |
| Service to Worker | X | — | X | X | — | X |
| Dispatcher View | X | — | X | X | — | X |

***Tabla 16: Matriz de consecuencias del uso de los patrones de diseño***[70]

[70] Figura basada en la información del estudio.

### *3.13.4.2 Patrones de la capa de negocio*.

La *capa de negocio* expone la lógica necesaria para que el usuario a través de la interfaz, interactúe con las funcionalidades de la aplicación.

La plataforma Java EE define el uso de componentes de negocio EJB para abstraer la lógica de negocio de otros problemas generales de las aplicaciones como la concurrencia, transacciones, persistencia y seguridad.

3.13.4.2.1 *Service Facade (Application Service).*

El Service Facade es la versión mejorada del patrón Application Service. Es un bean de sesión sin estado con una interfaz local de negocio que debe proporcionarse remotamente sólo si se va a utilizar fuera de la Máquina Virtual de Java (JVM), y no se inyecta en un Servlet, bean de respaldo, u otro componente web.

La motivación original del patrón Application Service sigue siendo útil: “Usted quiere centralizar la lógica de negocio a través de varios niveles de negocios, componentes y servicios” (Deepak Alur, 2003).

El patrón Service Facade debe atender las siguientes exigencias:

- Es necesaria una API de negocio, predefinida y fácil de consumir.
- El estado del componente después de la invocación del Service Facade debe ser consistente.
- La implementación detrás del patrón debe ser encapsulada.
- La interfaz del Service Facade debe utilizarse como un servicio remoto.
- La firma de los métodos expuestos debe ser estable y permanecer compatible, incluso si la implementación de los componentes cambia.

La tabla 17, ejemplifica un Service Facade con configuración estándar.

```
@Stateless
@Local (BookOrderingServiceLocal.class)
@Remote (BookOrderingServiceRemote.class)
@TransactionAttribute (TransactionAttributeType.REQUIRES_NEW)
public class BookOrderingServiceBean implements BookOrderingServiceLocal {
public void cancelOrder(String id) {
}
public void order(String isbn, String name) {
}
```

***Tabla 17: Implementación de un Service Facade con configuración estándar***[71]

### *Características del patrón*

- El Service Facade es el límite entre las capas de presentación y de negocio.
- Se compone de una interfaz local de negocio y de un bean de sesión sin estado. Todos los métodos inician una nueva transacción en cada llamada.
- Está diseñado para coordinar DAOs y servicios independientes, asegurándose de que tales servicios puedan ser reutilizados en otros contextos.
- Es anotado con TransactionAttributeType.REQUIRES_NEW. Esta configuración es heredada por todos los métodos.
- El atributo REQUIRES_NEW siempre inicia una nueva transacción y suspende la existente. Los métodos de la fachada son activados por el usuario final.
- Cada interacción entre el usuario y el sistema es una transacción de negocio que puede o no completarse.

[71] Ejemplo tomado del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

- Su responsabilidad principal continua siendo la composición de servicios independientes y reutilizables, pero en casos sencillos, se puede realizar directamente la lógica de negocio sin delegación a otros participantes.

### *Evolución del patrón*

En la plataforma J2EE, el patrón Application Service solía ser parte obligatoria en la implementación de aplicaciones J2EE. Se utilizó para aislar la capa de presentación de la capa de negocio. Tendía a desempeñar la función de controlador único y coordinaba los servicios existentes.

Los beans de entidad no eran separables, por lo que no se podía trabajar con ellos como parámetros o valores de retorno. La vista remota o local del Application Service así como su realización dependían de la API de los EJB 2.x.

### *Convenciones para el uso del patrón*

- El Service Facade reside en un paquete de Java con un nombre de dominio específico, por ejemplo, ordermgmt.
- La realización de la fachada (interfaz de negocio y la implementación del bean) reside en un subpaquete con el nombre fachada o frontera, por ejemplo, ordermgmt.facade ó ordermgmt.boundary.
- La interfaz de negocio es nombrada sin el sufijo local o remoto, por ejemplo, OrderService y OrderServiceBean.
- No es necesario utilizar el término facade en el nombre del bean. Para fines de identificación, se puede utilizar una anotación @ServiceFacade.
- El Service Facade siempre inicia una nueva transacción, no puede participar en una transacción en curso. Se despliega con el atributo de transacción REQUIRES_NEW.

### *Estrategias de implementación*

Existen algunas estrategias para la implementación del Service Facade, la tabla 18 menciona las más utilizadas.

| Estrategia | Características |
|---|---|
| Fachada CRUD | Es un Service Facade con la funcionalidad de un DAO.<br>Una fachada CRUD puede ser desplegada con una vista sin interfaz. |
| Fachada SOA | Es asíncrona y tiene metadatos ricos.<br>Todos los métodos son void.<br>Los mensajes entrantes son consumidos por un bean gestionado por mensajes y transformados en parámetros del Service Facade. |
| Fachada Ligera Asíncrona | Es una fachada SOA con EJB 3.1.<br>Se declarara los métodos como @Asynchronous y serán invocados en un subproceso.<br>Permite comunicación bidireccional. |

***Tabla 18: Características de las estrategias de implementación del patrón Service Facade***[72]

El uso de un patrón Service Facade afecta los siguientes puntos:

- Consistencia: Todos los métodos son transaccionales. La transacción se propaga a todos los objetos invocados en el ámbito de un Service Facade, el cual asegura la consistencia.
- Encapsulación: Este patrón desacopla de cierta forma, el cliente de los componentes que se encuentran detrás de la fachada.
- Rendimiento: La fachada hace una interfaz más gruesa lo cual es importante para la comunicación remota.

[72] Información obtenida del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

- Usabilidad: En lugar de invocar accesores, los métodos del Service Facade son más fluidos.
- Corte transversal: Un Service Facade es el único punto de entrada. Toda la comunicación se encamina a través de la fachada. Por lo tanto, es un lugar excelente para la implementación de inquietudes transversales como monitoreo, auditoría, o requisito previo de control.
- Desacoplamiento: Servicios independientes de granularidad fina, son coordinados por el Service Facade y pueden seguir siendo independientes y reutilizables en otro contexto.

La implementación del Service Facade se ve relacionada con los siguientes patrones de diseño: Application Service, Session Facade, Gateway, Service, DAO, DTO.

3.13.4.2.2 *Service (Session Facade).*

La intención original del Session Facade fue envolver beans de entidad y encapsular las iteraciones de granularidad fina con los objetos de negocio. La ejecución de consultas dinámicas fue delegada con frecuencia a un DAO.

En Java EE, un Service nuevo nombre del Session Facade, es procedimental y realiza actividades o subprocesos. Su objetivo principal es hacer que la construcción de la lógica de negocio sea reutilizable y más fácil de mantener. Debe satisfacer las siguientes necesidades:

- Los Services deben ser independientes entre sí.
- La granularidad de un Service es más fina que la de un Service Facade.
- El patrón Service no es accesible ni visible desde el exterior de la capa de negocio.

- Tiene como objetivo ser reutilizado desde otro componente o Service Facade.
- La ejecución de un Service debe ser siempre consistente. Cualquiera de sus métodos no genera efectos secundarios, por ser idempotentes, o tienen la capacidad de ser invocados en un contexto transaccional.

```
@Stateless
@Local(DeliveryService.class)
@TransactionAttribute(TransactionAttributeType.MANDATORY)
public class DeliveryServiceBean implements DeliveryService {
}
```

***Tabla 19: Ejemplo de implementación de un Service***[73]

### *Características del patrón*

- El Service no debe depender de la existencia de transacciones o comunicación remota; no es capaz de manejar llamadas remotas.
- Un Service es siempre local e incluye el atributo de transacción TransactionAttributeType.MANDATORY.
- Es un bean de sesión sin estado que puede ser desplegado con una interfaz de negocio local dedicada o con una vista sin interfaz. La elección depende de la complejidad del Service Facade que coordina Servicios.
- El lenguaje del Service realiza parte de la funcionalidad global del Service Facade, cuyas partes reutilizables e independientes son factorizadas fuera del Service.
- Un Service se debe ejecutar en el ámbito de una transacción existente.
- Todos los recursos que participan en el Service tienen acceso a la misma transacción.

[73] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

- Un Service utiliza Inyección de Dependencia para adquirir los recursos necesarios.
- Implementa la lógica de negocio y puede utilizar todos los recursos disponibles para su realización.

***Evolución del patrón***

El propósito principal del Session Facade fue la simplificación del acceso a los beans de entidad y crear granularidad gruesa. En contraste, el Service de Java EE 6 enfatiza los servicios específicos del dominio y es realmente independiente de la infraestructura de Java EE.

***Convenciones para el uso del patrón***

- Un Service es un bean de sesión local sin estado.
- Reside en un paquete de Java con nombre de dominio específico, por ejemplo: ordermgmt.
- La interfaz de negocio y la implementación del bean reside en un subpaquete con el nombre service o control, por ejemplo, ordermgmt.service ó ordermgmt.control.
- La interfaz de negocio debe su nombre a conceptos de negocio sin el sufijo obligatorio local o remoto, por ejemplo: OrderService y OrderServiceBean.
- No es necesario el uso del término Service en el nombre del bean. Se puede utilizar una anotación @Service.
- El Service siempre se invoca en el contexto de una transacción existente. Se despliega con el atributo de transacción Mandatory.
- Es ejecutado en el contexto de un Service Facade o Gateway.

El uso del patrón Service afecta los siguientes aspectos:

- Reutilización: Fomenta la reutilización interna de la lógica de granularidad fina. Puede ser compartido entre componentes y Service Facades.
- Consistencia: Se oculta detrás de una fachada y se ejecuta siempre en el contexto de la transacción ya existente.
- Complejidad: No todos los casos ameritan la introducción de una capa de Servicio adicional. La ejecución o delegación de servicios puede originar patrones Services o Service Facades vacíos. La construcción de un CRUD se conseguiría fácilmente con el uso exclusivo de un Service Facade.

***Estrategias de implementación***

Las estrategias SOA y Fachada Ligera Asíncrona del Service Facade, pueden ser aplicadas directamente a un Servicio, además las estrategias señaladas en la tabla 20.

| Estrategia | Características |
|---|---|
| Bean Stateless Session | La lógica de negocio se implementa como un bean de sesión sin estado con una interfaz local de negocio o con una vista sin interfaz.<br>La dependencia causada por anotaciones es mínima, la anotación puede ser removida completamente mediante descriptores de despliegue.<br>La sobrecarga del contenedor EJB resulta ser muy baja. |
| Estrategia POJO | El uso de un Service implementado como un POJO debe ser justificado.<br>El contenedor EJB no tiene conocimiento de POJOs puros.<br>Tampoco es capaz de manejar sus ciclos de vida ni para su monitoreo. |
| DAO híbrido | El EntityManager JPA es un DAO, pero es una buena idea encapsular consultas en un lugar específico, como un DAO o un Service.<br>La reutilización de consultas se puede lograr usando herencia.<br>Un Service puede heredar de una clase genérica DAO ya existente, o |

| | |
|---|---|
| | directamente del mismo Service, actuando como un DAO.<br>Esta estrategia es eficiente porque reduce y simplifica la implementación del Service y es fácil de mantener.<br>Los comportamientos dependientes del contexto pueden estar encapsulados en una superclase y ser constantemente reutilizados mediante los diferentes Servicios. |

***Tabla 20: Características de las estrategias de implementación del patrón Service***[74]

La implementación del Service se relaciona con los siguientes patrones de diseño: Application Service, Session Facade, Gateway, Service, DAO, DTO. Un Service es coordinado por Service Facades o Gateways, invoca DAOs y es capaz de producir DTOs.

3.15.4.2.3 *Persistent Domain Object (Business Object).*

La mayoría de aplicaciones J2EE se construyeron de forma procedimental. La lógica del negocio fue descompuesta en tareas y recursos, que fueron mapeados a servicios y entidades persistentes; lo cual funciona adecuadamente hasta que, en los objetos de dominio, un tipo específico de comportamiento deba ser implementado (Bien, 2011).

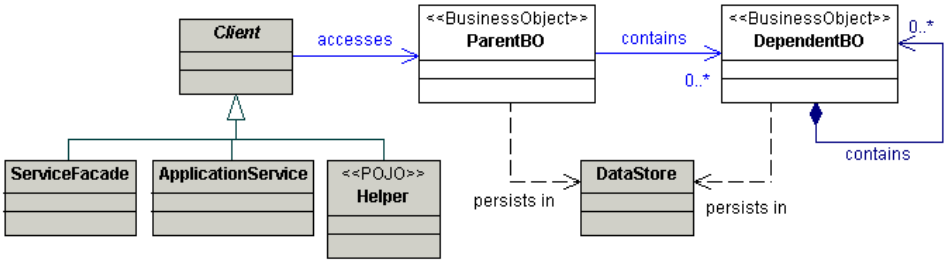


***Figura 19: Diagrama de clases del patrón Business Object J2EE***[75]

[74] Información obtenida del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

[75] Figura tomada del sitio web del libro Core J2EE Patterns (WWW47).

El patrón Persistent Domain Object u Objeto de Dominio Persistente (PDO) es una entidad persistente. Las relaciones y los cambios en el estado se sincronizarán automáticamente con la persistencia al final de la transacción. Debe cumplir las siguientes exigencias:

- Su lógica de negocio es compleja.
- Las reglas de validación son objetos de dominio relacionados y sofisticados.
- El modelo conceptual se puede derivar de los requerimientos y es mapeado a objetos de dominio.
- Los objetos de dominio deben persistir en una base de datos relacional.

***Características del patrón***

- El PDO presenta y desarrolla la lógica de negocio, lo más sencilla y explícitamente posible.
- El estado del PDO es conservado en la base de datos por el EntityManager.
- Un PDO debe ser creado y mantenido por un Service, Service Facade o Gateway.
- Si se transfiere un PDO a través de la red, se vuelve serializado y se desprende.

***Evolución del patrón***

La persistencia CMP de J2EE no admite herencia o consultas polimórficas. Java EE, juntamente con JPA, introdujo herencia y consultas polimórficas a la capa de persistencia. Las entidades JPA son clases Java anotadas con restricciones mínimas, por lo que los enfoques orientados a objetos y patrones pueden ser aplicados en la capa de persistencia.

### *Convenciones para el uso del patrón*

- Los PDOs son entidades JPA, con énfasis en la lógica de dominio.
- Reside en un subpaquete con el nombre domain o entity, por ejemplo, ordermgmt.domain u ordermgmt.entity.
- El nombre del objeto del dominio se deriva del dominio destino.
- Los getters y setters no son obligatorios, deben ser utilizados sólo en casos justificados.
- El comportamiento del modelo también cambia el estado de los objetos del dominio.
- Todos los métodos son nombrados de manera fluida.
- No es necesario usar las siglas PDO en el nombre de la entidad. Para fines de identificación se puede utilizar una anotación @PDO.

```
@Entity
public class BulkyItem extends OrderItem{
public BulkyItem() {
}
public BulkyItem(int weight) {
super(weight);
}
@Override
public int getShippingCost() {
return super.getShippingCost() + 5;
}}
```

***Tabla 21: Ejemplo de implementación de un PDO***[76]

### *Estrategias de implementación*

Sin importar qué tipo de estrategia se elija, los métodos de negocio PDO siempre deben ejecutarse en un estado asociado.

[76] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

El acceso a los PDOs debe ser por referencia, que implica la ejecución de la capa de presentación y de PDOs en la misma JVM. Esto es habitual en clientes web, sin embargo, en un entorno de cliente rico, se requiere una ejecución en proceso del EntityManager dentro de la capa de presentación. La tabla 22, señala las estrategias que pueden ser utilizadas para implementar un objeto de dominio persistente.

| Estrategia | Características |
|---|---|
| DSL/Prueba/ Manejador de Dominio | El estado está completamente encapsulado y el empleo de accesores no es muy frecuente.<br>La información que necesita un PDO es transformada y proporcionada manualmente. |
| Fat PDO | La inyección del EntityManager no es compatible en entidades persistentes.<br>El desarrollador deberá asociar la entidad recién creada con un objeto que ya esté vinculado o invocar EntityManager#persist.<br>El cliente tiene acceso a los PDOs a través del Service Facade. |
| PDO Enlazado a Datos | Variante especializada del PDO que trata el intercambio entre la interfaz fluida o enfoque administrado por el dominio y un procedimiento DTO.<br>Dispone de descriptores de acceso, que están destinados a la sincronización automática con los componentes de la capa de presentación. |

***Tabla 22: Características de las estrategias de implementación del PDO***[77]

El uso del patrón Persistent Domain Object ocasiona los siguientes efectos:

- Usabilidad: La API del PDO es limpia, concisa, fácil de usar y debe ser transparente para los expertos del dominio.
- Experiencia en el dominio: Los conceptos del dominio deben reflejarse directamente en el PDO.

[77] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

- Responsabilidades básicas esenciales: Un PDO debe reflejar sólo conceptos clave y no debe plasmar aspectos transversales.
- Extensibilidad: Los PDOs se pueden ampliar fácilmente usando mecanismos orientados a objetos y herencia.
- Capacidad de transferencia de datos: Un PDO se puede transferir fácilmente entre capas, incluso remotamente como objetos separados. El único requisito es la implementación de la interfaz java.io.Serializable. La lógica de negocio puede ser invocada en un estado separado.

La implementación del Persistent Domain Object se relaciona con los siguientes patrones de diseño: Composite Entity, Domain Store, Business Object, Data Transfer Object.

3.13.4.2.4 *Gateway*.

Un Gateway proporciona un punto de entrada a la raíz de los PDOs. Debe satisfacer las siguientes exigencias:

- Los PDOs son accesibles localmente, por referencia.
- Los PDOs ya están encapsulados, no hay necesidad de estratificación adicional y abstracción.
- La aplicación es la dueña de la base de datos, por lo que puede ser cambiada en la demanda.
- Los cambios realizados en la interfaz de usuario pueden afectar a la base de datos.
- La aplicación es interactiva. La lógica de validación depende del estado de un objeto.
- Los usuarios deciden cuando sus cambios son almacenados y cuando no.
- La lógica de negocio no se puede expresar en forma orientada a servicios, con servicios autocontenidos, independientes y atómicos.

***Características del patrón***

- El Gateway expone los PDOs directamente a la presentación, por lo que este patrón, pierde el control sobre PDOs y transacciones.
- Un Gateway puede mantener PDOs unidos con un EntityManager declarado como PersistenceContext.EXTENDED.
- La implementación de un Gateway es un bean de sesión con estado con un EntityManager extendido, y, todas las transacciones entrantes van a ser suspendidas con el atributo TransactionAttributeType.NOT_SUPPORTED en el nivel de clase.
- Puede conservar un estado específico del cliente, que es también una referencia a un PDO. Esto se consigue mediante la inyección del Gateway a un componente web con estado.
- El ciclo de vida del Gateway es dependiente del ciclo de vida de la HttpSession.
- Mantiene el EntityManager abierto entre llamadas al método.
- Proporciona un punto de sincronización definido.
- Acceso a PDOs por referencia local. Esto requiere que se ejecute el contenedor EJB en la misma JVM del contenedor web.

Depende de algunas cualidades PDO, como:

- Los PDOs deben ser diseñados para acceder directamente en la presentación. Su funcionalidad debe estar bien encapsulada.
- Los métodos públicos PDO deben dejar el objeto de dominio en un estado consistente. Esto difícilmente se puede obtener con simples getters y setters.

### *Convenciones para el uso del patrón*

- Un Gateway reside en un paquete de Java con un nombre específico, por ejemplo, ordermgmt.
- Ocupa un subpaquete con el nombre facade o boundary, por ejemplo, ordermgmt.facade ó ordermgmt.boundary. Se encuentra en el mismo subpaquete que un Service Facade.
- Un Gateway es un bean local de sesión con estado con el atributo de transacción NOT_SUPPORTED o el evento NEVER en el nivel de clase.
- Los métodos dedicados, mayoritariamente vacíos, comienzan una nueva transacción con la configuración REQUIRES_NEW y limpian el estado transitorio de la base de datos.
- El Gateway es desplegado con una interfaz local de negocio simple, es posible también la optimización de la vista sin interfaz.

La implementación del Gateway se relaciona con los siguientes patrones de diseño: Composite Entity, Domain Store, Business Object, Data Transfer Object, Service Facade, Service.

```
@Stateful
@TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
@Local(OrderGateway.class)
public class OrderGatewayBean implements OrderGateway{
private Load current;
public Load getCurrent() {
return current;
}
@TransactionAttribute(TransactionAttributeType.REQUIRES_NEW)
@Remove
public void closeGate(){
}}
```

***Tabla 23: Ejemplo de implementación del patrón Gateway***[78]

[78] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Estrategias de implementación*

La ejecución en proceso del patrón Gateway es común en todas las estrategias. La tabla 24, muestra las características principales de cada una de ellas.

| Estrategia | Características |
|---|---|
| Server-Side Gateway | El Gateway es desplegado como un bean de sesión con estado y es accedido por el contenedor web mediante referencia.<br>Permite omitir la interfaz de negocio. |
| Gateway RIA | El Gateway se ejecuta directamente en la máquina virtual del cliente.<br>Puede acceder a una base de datos incorporada o una base de datos remota.<br>Requiere tener al menos un constructor por defecto y un campo anotado con @PersistenceContext. |
| Gateway Híbrido | El Gateway puede ser desplegado en paralelo en el cliente y servidor.<br>El despliegue directo del Gateway al cliente hace casi imposible la exposición de servicios reutilizables al exterior.<br>Un despliegue en el lado del servidor ocasiona la fácil exposición de las funcionalidades existentes a través de REST, SOAP, RMI o IIOP. |

***Tabla 24: Características de las estrategias de implementación del Gateway***[79]

El uso del patrón Gateway produce los siguientes efectos:

- Escalabilidad: Es altamente dependiente del tamaño de la memoria caché y la duración de la sesión. Este patrón no se escala tan bien como un servicio sin estado.

[79] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

- Funcionamiento: Cuando el usuario necesita acceso frecuente a los mismos datos, una arquitectura con estado podría proporcionar un mejor rendimiento. Los PDOs se cargan sólo una vez durante toda la sesión y pueden ser accedidos localmente por referencia.
- Consistencia: La introducción del patrón Gateway implica automáticamente el uso de estrategias de bloqueo optimista[80].
- Mantenibilidad: Un Gateway simplifica la arquitectura y hace capas intermediarias prolijas. Pero puede aumentar la complejidad de una aplicación orientada a servicios, donde los PDOs representan un subconjunto de la lógica de negocio, y, las interfaces de los sistemas externos operan con DTOs que están separados por definición. Además, la exposición directa del estado interior y del comportamiento con PDOs encapsulados y conectados, hacen que el desarrollo de la lógica de presentación sea más compleja y de arquitectura frágil.
- Productividad: Los PDOs son expuestos directamente a la capa de presentación y están disponibles para el enlace de datos declarativos, o pueden ser manipulados directamente en el controlador. Cada ampliación en la lógica de dominio está disponible de inmediato en la presentación.

3.13.4.2.5 *Fluid Logic.*

Los algoritmos que cambian a menudo requieren re-compilación e incluso redespliegue de toda la aplicación. El patrón Fluid Logic ó Lógica de Fluidos, debe satisfacer los siguientes requisitos:

- El desarrollador desea ejecutar partes de la lógica de negocio en forma dinámica.

[80] **Bloqueo optimista,** no bloquea los registros que se van a actualizar y asume que los datos que están siendo actualizados no van a cambiar desde que se han leído (Gracia).

- Los algoritmos que con frecuencia cambian deben ser aislados y se cargan dinámicamente, sin afectar el resto de la aplicación.
- El código afectado cambia estructuralmente, la compilación en tiempo de despliegue no es suficiente.
- La creación de un intérprete personalizado es demasiado gravoso y difícil de mantener.

**Convenciones para el uso del patrón**

El patrón Fluid Logic es un Service específico por lo cual, todas las convenciones descritas en el patrón Service, se aplican también a la Lógica del Fluido.

```
import javax.annotation.PostConstruct;
import javax.ejb.Stateless;
import javax.script.Bindings;
import javax.script.ScriptContext;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
@Stateless
public class Calculator {
private static final String ENGINE_NAME = "JavaScript";
private ScriptEngine scriptEngine = null;
@PostConstruct
public void initScripting() {
ScriptEngineManager engineManager = new ScriptEngineManager();
this.scriptEngine = engineManager.getEngineByName(ENGINE_NAME);
if (this.scriptEngine == null) {
throw new IllegalStateException("Cannot create… " + ENGINE_NAME);
} }
```

***Tabla 25: Ejemplo de implementación del Fluid Logic***[81]

[81] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Evolución del patrón*

En el período de J2EE 1.x, el scripting en las aplicaciones fue considerado inapropiado. En JEE, la lógica de fluidos es fácilmente reemplazable y puede ser cambiada en un sistema en ejecución sin redespliegue. Debido a que los scripts son cargados dinámicamente, pueden ser almacenados y gestionados en una base de datos o descargados desde la red.

### Estrategias de implementación

No hay límites en la integración del scripting y por lo tanto existen numerosas estrategias. El patrón de Lógica de Fluidos se puede utilizar en todos los niveles, capas y componentes para diferentes propósitos. Todas las variaciones apuntan a un objetivo común: la integración transparente entre Java y las secuencias de comandos. El uso del patrón Fluid Logic afecta los siguientes puntos:

- Rendimiento: Afectará el rendimiento de la aplicación.
- Mantenibilidad: La intención del patrón Fluid Logic es lograr que la lógica cambiante sea más fácil de mantener. El impacto del scripting en los proyectos de Java depende de las habilidades del desarrollador y de los casos en que se use.
- Portabilidad: Los scripts se ejecutan dentro de la JVM y son portables entre las JVM y los servidores de aplicaciones.
- Flexibilidad y agilidad: Los scripts se pueden modificar, cargar, y reemplazar en tiempo de ejecución, sin tener que volver a implementar la aplicación. El script incluso, no tiene que ser desplegado en el interior del EAR, WAR, o EJB JAR.

- Seguridad: A pesar de los scripts son ejecutados dentro de la JVM y se ejecutan en un entorno protegido por el SecurityManager, la introducción de scripting ofrece un potencial e impredecible agujero de seguridad debido a que todavía se podría introducir un script malicioso que comprometa la consistencia de la lógica del negocio.

3.13.4.2.6 *Paginator y Fast Lane Reader.*

El patrón original Fast Lane Reader, fue diseñado para:

El patrón de diseño Fast Lane Reader proporciona una manera más eficiente de acceder a datos tabulares de sólo lectura. Un componente de Fast Lane Reader accede directamente a datos persistentes utilizando componentes JDBC, en lugar de utilizar beans de entidad. El resultado es un mejor rendimiento y menos codificación, ya que el componente representa los datos en una forma que está más cercana al cómo estos datos son usados (Sun Microsystems, Inc, 2002).

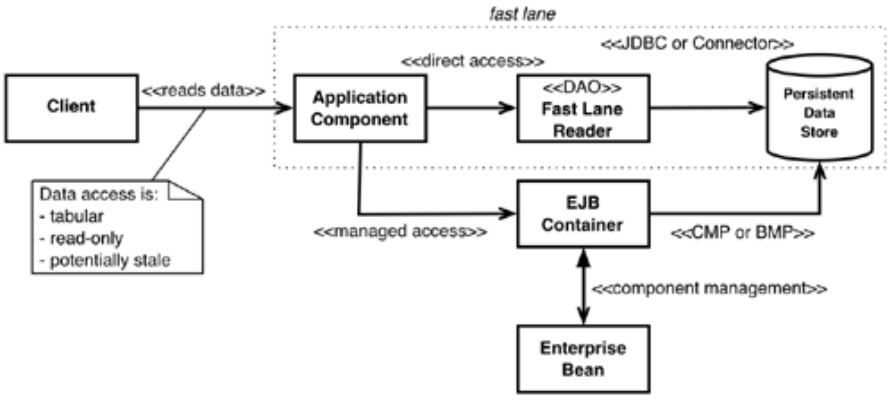


***Figura 20: Diagrama de clases del patrón Fast Lane Reader[82]***

[82] Figura tomada del Catálogo BluePrints (Sun Microsystems, Inc, 2002).

El patrón Fast Lane Reader es un Service Facade especializado o en aplicaciones más complejas, un Service reutilizable. Debe satisfacer las siguientes exigencias:

- El desarrollador debe iterar sobre una gran cantidad de datos.
- Los datos no se pueden cargar a la vez en el cliente y deben ser almacenados en la caché del servidor.
- El cliente está interesado solamente en algunos atributos de la entidad.
- Los datos son accedidos principalmente en forma de sólo lectura.

**Convenciones para el uso del patrón**

El patrón Fast Lane Reader es una forma específica de un Service Facade o Service, por lo que las convenciones correspondientes a dichos patrones, se pueden aplicar directamente a este patrón de diseño.

```
@Stateless
public class CustomerQueryPageOrEverythingBean{
@PersistenceContext
EntityManager em;
private static int MAX_PAGE_SIZE = 1000;
public List<Customer> getAllCustomers(int maxResults){
if(maxResults == -1 || maxResults > MAX_PAGE_SIZE)
maxResults = MAX_PAGE_SIZE;
else
maxResults +=1;
Query query = this.em.createNamedQuery(Customer.ALL);
query.setMaxResults(maxResults);
return query.getResultList();
}}
```

***Tabla 26: Ejemplo de implementación del Fast Lane Reader***[83]

[83] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Características del patrón*

- El patrón Fast Lane Reader o lector de vía rápida, es accedido localmente y utilizado directamente por el cliente.
- Puede ser considerado como un Service Facade ligero e independiente y, en la mayoría de casos, opera directamente en el EntityManager.
- No hay necesidad de acceder a un DAO o Service para obtener acceso simplificado a la persistencia porque el EntityManager es simple.

El Paginator utiliza el EntityManager. En raros casos, podría depender de un Service existente o una implementación DAO. Es usado directamente por sus clientes, por ejemplo, beans de respaldo, servicios RESTful, Servlets. Un DAO facilita el acceso Fast Lane Reader mediante la encapsulación de la lógica necesaria para recuperar los datos a partir de un recurso.

### *Estrategias de implementación*

El principio de todas las estrategias es exactamente el mismo, la iteración del lado del servidor en una colección grande de objetos persistentes. La tabla 27 muestra las características de estas estrategias.

| Estrategia | Características |
|---|---|
| Paginator and Value List Handler | El Paginator implementa la interfaz java.util.Iterator<br>Su creación requiere de la implementación de un bean de sesión con estado y utiliza el EntityManager con el tipo predeterminado PersistenceContextType.TRANSACTION.<br>La API de EJB y la existencia de un bean de sesión implementando la interfaz, están totalmente ocultos. |
| Page or Paginator Everything | Esta estrategia es una forma simplificada del empleo del Paginator clásico.<br>No utiliza el java.util.Iterator, solo comprueba si la primera página contiene el resultado completo, si no lo tiene, pedirá al usuario que vuelva a ejecutar la consulta y devolverá un número todavía limitado de objetos o refinará la consulta.<br>Se implementa como un bean de sesión sin estado.<br>No hay necesidad de mantener el índice actual u objetos de caché. |
| Table View | Provee una forma eficiente de iterar sobre un conjunto de entidades relacionadas devolviendo una vista distinta al cliente.<br>Puede ser aplicada a todas las entidades JPA, proporcionando mayor flexibilidad y facilitando la refactorización de la base de datos. |
| Live Cursor and Fast Lane Reader | Con EJB 3, se puede obtener acceso directo a la java.sql.Connection, sin el uso del EntityManager.<br>Los objetos se construirán una vez y podrán reutilizarse varias veces.<br>Se itera sobre los datos en la base de datos sin convertir las filas dentro de los objetos, ahorrando tiempo y recursos. |

***Tabla 27: Características de las estrategias de implementación del Fast Lane Reader***[84]

[84] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

El uso del Paginator y del patrón Fast Lane Reader produce:

- Rendimiento: El rendimiento es mejor cuando se utiliza el Paginator, en lugar de retornar un gran conjunto resultado a la vez.
- Escalabilidad: Las implementaciones del Paginator con estado, deben mantener el estado y ser replicado en otros nodos del clúster.
- Robustez: El Paginator retorna segmentos de un conjunto de resultados a sus clientes y evita que se quede sin memoria.
- Consistencia: Las estrategias del Paginador guardan en caché al menos la página actual. Las entidades permanecen unidas, no se actualizan automáticamente desde la base de datos
- Portabilidad: El patrón Paginator es portable a través de servidores de aplicaciones. El Fast Lane Reader es la única estrategia que utiliza SQL nativo directamente y puede depender de la base de datos particular.

3.13.4.2.7 *Patrones retirados.*

Los patrones que se detallan a continuación están retirados de las principales aplicaciones de Java EE 6, pero podrían ser utilizados para fines especiales o durante la migración de J2EE a Java EE 6.

3.13.4.2.7.1 Service Locator.

Fue un patrón obligatorio en aplicaciones J2EE. Todos los recursos y componentes debían situarse primero.

Usar un objeto Service Locator para abstraer todo el uso de JNDI y para ocultar las complejidades de creación del contexto inicial, el objeto de búsqueda EJB origen y el objeto EJB de re-creación. Varios clientes pueden reutilizar el objeto Service Locator para reducir la complejidad del código, proporcionar un único punto de control, y mejorar el rendimiento al proveer un servicio de caché (Sun Microsystems, Inc, 2002).

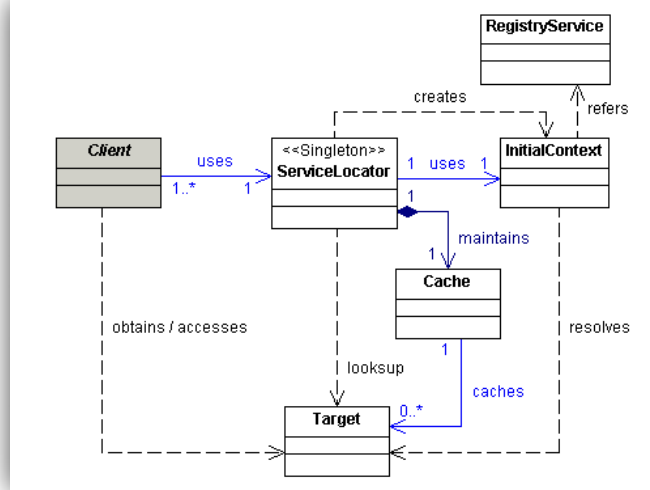


***Figura 21: Diagrama de clases del patrón Service Locator***[85]

### *Razones para el retiro del patrón*

- La Inyección de Dependencia está disponible en la mayoría de los componentes de Java EE. Las búsquedas JNDI ya no son necesarias para acceder a otros beans de sesión o recursos.
- La creación de una interfaz inicial es opcional, y por consiguiente la creación de interfaces remotas y locales.
- Desde EJB 3.0, el bean de sesión puede ser accedido con Context#lookup.

[85] Figura tomada del sitio web del libro Core J2EE Patterns (WWW47).

- La complejidad del código de infraestructura fue reducida en gran medida por la especificación EJB 3.0. Se debe utilizar la Inyección de Dependencia siempre que sea posible y sólo en casos excepcionales, la implementación de un Service Locator genérico.
- En determinados casos se puede utilizar un Service Locator especializado llamado BeanLocator.

*3.13.4.2.7.2 Composite Entity.*

"Utilizar Composite Entity para modelar, representar y gestionar un conjunto de objetos persistentes relacionados entre sí en lugar de representarlos como beans de entidad individuales de granularidad fina. Un bean Composite Entity representa un esquema de objetos" (Sun Microsystems, Inc, 2002).

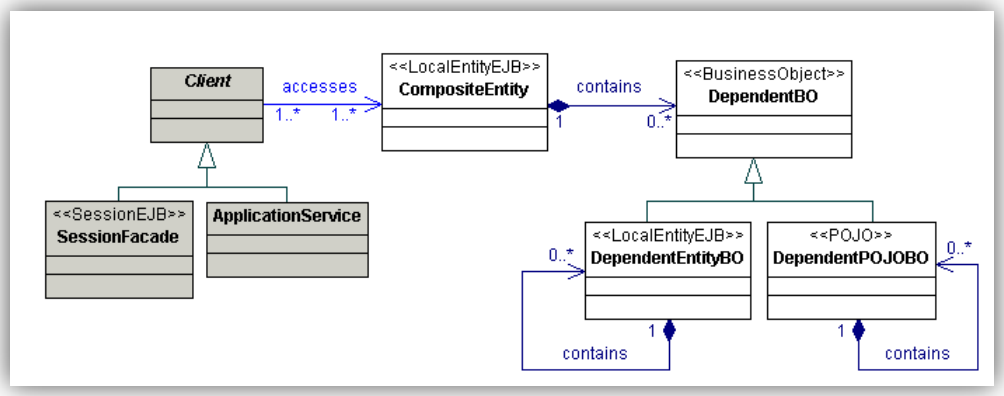


***Figura 22: Diagrama de clases del patrón Composite Entity[86]***

[86] Figura tomada del sitio web del libro Core J2EE Patterns (WWW47).

***Razones para el retiro del patrón***

- La persistencia CMP 2.1 no soporta perfectamente relaciones. Las entidades CMP tenían interfaces de inicio que fueron utilizadas de manera similar al EntityManager. Con la llegada de JPA la implementación de las relaciones se tornó natural.
- Las Entidades JPA son objetos de dominio que son persistentes. Se puede aplicar cualquier patrón que se desee sin ninguna sobrecarga.
- Las entidades JPA son orientadas a objetos por defecto. El patrón Composite Entity en Java EE fue degradado a un tradicional Composite del catálogo GoF.

*3.13.4.2.7.3 Value Object Assembler.*

El Value Object Assembler fue un patrón dedicado a fusionar, transformar o extraer datos desde diferentes fuentes de datos.

"Utilizar un Transfer Object Assembler para construir el modelo o submodelo requerido. El Transfer Object Assembler utiliza Transfer Objects para recuperar datos desde varios objetos de negocio y otros objetos que definen el modelo o parte del modelo" (Sun Microsystems, Inc, 2002).

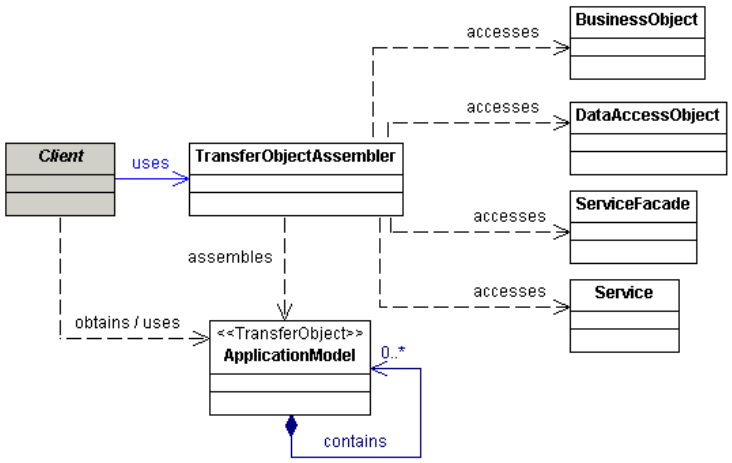


***Figura 23: Diagrama de clases del patrón Value Object Assembler***[87]

### *Razones para el retiro del patrón*

- El EntityManager implementa parte de la intención original del Value Object Assembler: es capaz de retornar un submodelo desde un esquema de entidades interconectadas.
- La creación de submodelos y la conversión de entidades adjuntas dentro de Transfer Objects, es responsabilidad del Service Facade o Service. No hay necesidad de introducir un componente específico o patrón para implementar la conversión.
- En la mayoría de casos, se podría deshacer del uso de Transfer Objects dedicados y pasar entidades separadas y unidas entre capas o niveles.

#### *3.13.4.2.7.4 Business Delegate.*

Los clientes J2EE fueron expuestos directamente a la API de J2EE y a complejidad.

[87] Figura tomada del sitio web del libro Core J2EE Patterns (WWW47).

Utilizar un Business Delegate para reducir el acoplamiento entre la capa de presentación, clientes y servicios de negocios. El Business Delegate oculta los detalles de implementación del servicio de negocio, como búsquedas y el acceso a detalles de la arquitectura EJB (Sun Microsystems, Inc, 2002).

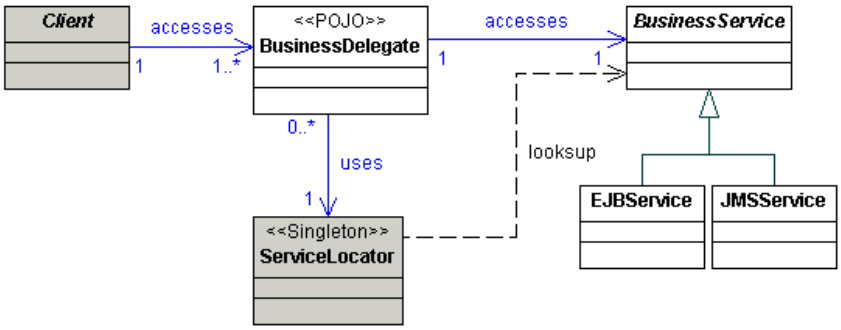


***Figura 24: Diagrama de clases del patrón Business Delagate***[88]

### *Razones para el retiro del patrón*

- La mayoría de las excepciones EJB 2.1 fueron revisadas. El cliente EJB debía capturar excepciones y fue contaminado con la API de EJB 2.1. Con EJB 3.0 las excepciones chequeadas son opcionales, la interfaz de negocio es idéntica a la interfaz externa del Business Delegate.
- El Business Delegate hacía uso del Service Locator para buscar la interfaz inicial y creaba internamente la interfaz local o remota. En EJB 3.0, las interfaces de origen son opcionales, las interfaces de negocios pueden ser recuperadas directamente desde JNDI.
- Los Business Delegates se utilizaron también para separar la lógica de negocio de la presentación. Esto ya no es necesario, porque las interfaces de negocio pueden ser inyectadas directamente en la mayoría de los componentes de presentación.

[88] Figura tomada del sitio web del libro Core J2EE Patterns (WWW47).

### *3.13.4.2.7.5 Domain Store.*

Usar un Domain Store para persistir de forma transparente un modelo de objetos. A diferencia de la persistencia gestionada por el contenedor J2EE y la persistencia de bean gestionado, que incluyen código de persistencia de apoyo en el modelo de objetos, el mecanismo de persistencia del Domain Store es independiente del modelo del objeto (WWW47).

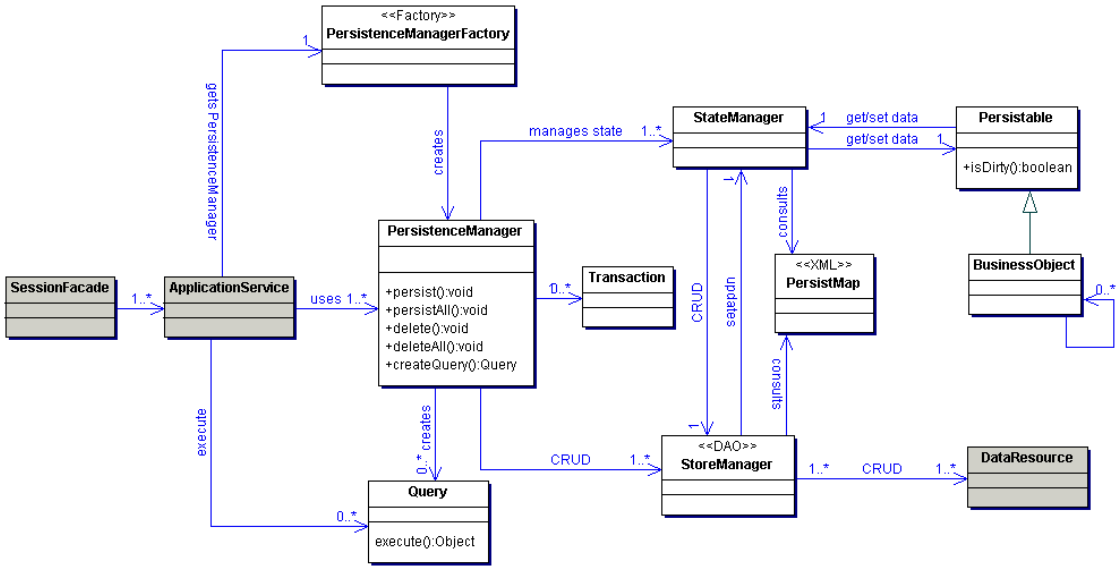


***Figura 25: Diagrama de clases del patrón Domain Store***[89]

## *Razones para el retiro del patrón*

El EntityManager puede ser considerado como una implementación estandarizada del patrón Domain Store. Es el patrón Domain Store.

[89] Figura tomada del sitio web del libro Core J2EE Patterns (WWW47).

*3.13.4.2.7.6 Value List Handler.*

El CMP no proporcionó ninguna semántica para la iteración de datos o separación. El patrón Value List Handler fue introducido para corregir esta limitación:

"Utilizar un Value List Handler para controlar la búsqueda, almacenar en caché los resultados, y presentar los resultados al cliente en un conjunto de resultados cuyo tamaño y recorrido cumple con los requisitos del cliente" (Sun Microsystems, Inc, 2002).

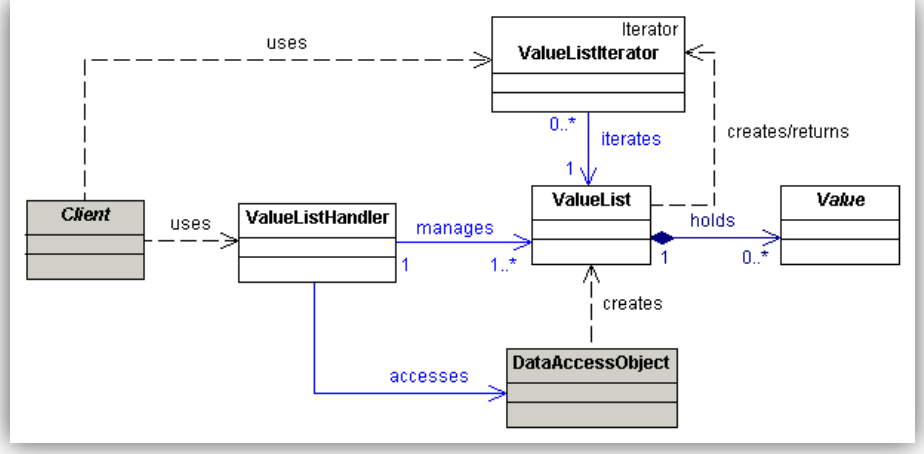


***Figura 26: Diagrama de clases del patrón Value List Handler***[90]

### *Razones para el retiro del patrón*

- El Value List Handler fue el responsable de la conversión de instancias CMP dentro de Transfer Objects. Desde la introducción de JPA, las entidades se pueden separar fácilmente sin esfuerzo adicional.

[90] Figura tomada del sitio web del libro Core J2EE (WWW47).

- La implementación del Value List Handler es bastante trivial, porque es la aplicación del patrón Iterator[91] clásico.
- El Value List Handler se convirtió en una estrategia del patrón Paginator.

### *3.13.4.3 Patrones de la capa de integración.*

La *capa de integración* agrupa componentes de conexión con otros sistemas, conectores con servicios, servicios de mensajería y otros. En esta capa se numeran los siguientes patrones:

#### *3.13.4.3.1 Data Access Object.*

"La motivación para el uso de este patrón fue el desacoplamiento de la lógica de negocio, de la construcción concreta de los mecanismos del almacén de datos" (Real World Java EE Patterns – Rethinking Best Practices, 2009).

Las aplicaciones todavía necesitan encapsular los recursos heredados, pero ya no es una regla. El patrón DAO debe satisfacer los siguientes requerimientos:

- Se debe tener acceso a una fuente de datos heredada o a recursos.
- La abstracción de acceso a datos debe conservarse comprobable y fácil de burlar.
- La aplicación está orientada a servicios: la lógica del negocio y el acceso a datos se separan.
- Se debe tratar con fuentes de datos heredados no estándar.
- Las consultas demasiado complejas, se deben mantener en un lugar específico.

[91] Iterator, permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.

### *Características del patrón*

- Un patrón DAO se reduce, en un entorno Java EE 6, a una interfaz y una clase, con anotaciones EJB específicas.
- El bean de sesión DAO es anotado con @TransactionAttribute, por lo que todos los métodos DAO pueden ser invocados en una transacción existente.
- Un DAO debe ser invocado desde un componente de la lógica del negocio y no directamente desde la presentación.
- En Java EE 6, el uso de un DAO dedicado es opcional, puede ser reemplazado con un EntityManager inyectado en un Service.

### *Evolución del patrón*

En un ambiente EJB 3 de Java EE 6, se puede utilizar JPA y SQL nativo para recuperar objetos persistentes, DTOs y tipos de datos primitivos desde la base de datos. La JPA viene con el EntityManager, que se puede considerar como una implementación genérica del patrón DAO, proporcionando una funcionalidad genérica de acceso a datos y puede ser inyectado directamente a cualquier bean de sesión y a los servicios existentes. Sus métodos son muy similares a las implementaciones DAO (CRUD).

### *Convenciones para el uso del patrón*

La implementación concreta de la interfaz CrudService va a ser inyectada en el cliente por el contenedor. Todas las estrategias y variaciones de DAO sólo se componen de una interfaz de negocio y su implementación bean. Una inyección directa de la implementación DAO hace que las pruebas fuera del contenedor sean más difíciles.

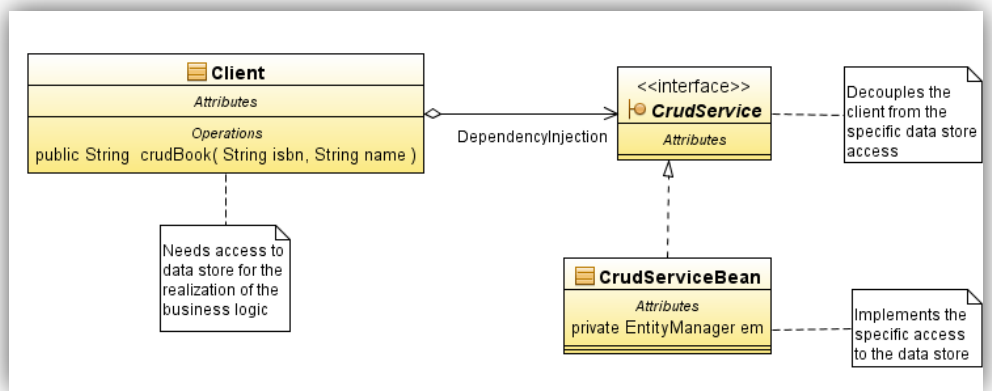


***Figura 27: Estructura del patrón DAO***[92]

- El cliente DAO se implementa como un bean de sesión con o sin estado que tiene acceso a la funcionalidad DAO.
- El DAO siempre se invoca en el contexto de una transacción activa y es inyectado en el cliente por el contenedor.
- Se implementa como un bean de sesión sin estado y es el encargado de la encapsulación de APIs y tecnologías patentadas.
- Un DAO genérico se implementa una vez y es reutilizado desde varios componentes; por lo tanto, debe ser situado en un paquete genérico.
- Proporciona un adecuado acceso al almacén de datos.
- Su principal responsabilidad es la gestión de los recursos y la conversión de datos en objetos de dominio de nivel superior.

**Estrategias de implementación**

Todas las estrategias tienen como objetivo la simplificación del acceso a datos y el desacoplamiento pragmático del almacén de datos. La tabla 28 señala las estrategias que se pueden utilizar.

[92] Figura tomada del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

| Estrategia | Características |
|---|---|
| Generic DAO | El DAO genérico es adecuado para la gestión de datos y simples casos CRUD.<br>Los métodos create y update no son void, pero tienen el mismo tipo de retorno como su parámetro.<br>La implementación de la interfaz es un bean de sesión local sin estado con el ajuste de transacción MANDATORY.<br>La interfaz CrudService es expuesta con la anotación @Local. |
| Domain specific DAO | Un DAO específico de dominio, realiza una funcionalidad específica y accede a DAOs genéricos para la reutilización de la funcionalidad básica.<br>Esta variante debe ser desplegada una vez para un tipo de entidad, permitiendo que una instancia de un DAO genérico gestione entidades diferentes. |
| Attached result DAO | Esta estrategia retorna objetos activos que siguen conectados al almacén de datos.<br>Es establecida para un DAO implementado como un bean de sesión sin estado y un EntityManager inyectado.<br>Si el DAO reutiliza la transacción que encierra, todas las entidades devueltas seguirán siendo administradas.<br>Siempre devuelve la misma referencia a una entidad para una clave primaria determinada. |
| Detached result DAO | En esta estrategia los resultados del método son desprendidos.<br>Su responsabilidad principal es la conversión entre un conjunto de datos orientado a recursos propietarios y POJOs más convenientes. |
| Back-end Integration DAO | La estructura y la construcción de este variante son semejantes a un DAO estándar pero interactúa con los servicios existentes back-end en lugar de acceder a la base de datos.<br>Interactúa con el proveedor específico y con la funcionalidad generada. |
| Abstract DAO | La lógica reutilizable de acceso a datos se puede heredar de una implementación de DAO abstracto; en lugar de delegarla en un DAO específico, los métodos DAO estarían disponibles de inmediato. |

***Tabla 28: Características de las estrategias de implementación del patrón DAO***

```
public interface CrudService {
<T> T create(T t);
<T> T find(Object id,Class<T> type);
<T> T update(T t);
void delete(Object t);
List findByNamedQuery(String queryName);
List findByNamedQuery(String queryName,int resultLimit);
List findByNamedQuery(String namedQueryName, Map<String,Object> parameters);
List findByNamedQuery(String namedQueryName, Map<String,Object> parameters,int
resultLimit);
}
```

***Tabla 29: Ejemplo de implementación del patrón DAO***[93]

El uso del patrón Data Access Object genera las siguientes consecuencias:

- Complejidad adicional: Un DAO introduce al menos dos elementos adicionales, una interfaz y su implementación que deben ser desarrolladas, documentadas y mantenidas.
- No es DRY: El EntityManager ya encapsula las diferentes implementaciones JPA. Un DAO no debe ser utilizado para la implementación de aplicaciones CRUD simples o como un envoltorio alrededor de un EntityManager.
- Encapsulación de la funcionalidad reutilizable del acceso a datos: La introducción de DAOs puede ser útil en el caso de una función adicional, la lógica específica puede ser encapsulada en un lugar central. Este enfoque minimiza la duplicación de código y mejora la capacidad de mantenimiento.

[93] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

El patrón DAO es un arreglo genérico de los patrones Domain Store, Fast Lane Reader o Value List Handler. Un DAO es usado por un Service o Service Facade y podrá utilizar JCA Genérico para acceder a recursos propietarios heredados.

*3.13.4.3.2 Transfer Object and Data Transfer Object.*

El planteamiento original del problema de un Objeto de Transferencia de Datos (DTO) o un Objeto de Transferencia (TO) fue: "Usted desea transferir múltiples elementos de datos en más de un nivel" (Deepak Alur, 2003).

Para resolver este problema el desarrollador debía copiar los estados de las entidades en un Transfer Object, por ser una estructura remotamente transferible.

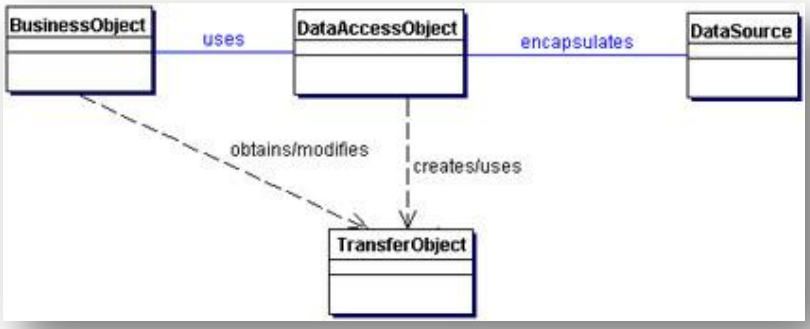


***Figura 28: Diagrama de clases del patrón DTO J2EE[94]***

En Java EE, los TOs son utilizados para proporcionar vistas específicas de los consumidores a la capa de persistencia y para mantener estables las interfaces externas. Deben solventar los siguientes requerimientos:

[94] Figura tomada del sitio web del libro Core J2EE Patterns (WWW47).

- Los cambios realizados en un cliente no deben afectar a los otros.
- La transferencia de entidades profundamente interconectadas debe ser optimizada.
- Se requieren vistas diferentes para un modelo de dominio único.
- Los datos de recursos propietarios y sistemas de información deben ser transferidos al cliente.
- El desarrollador debe comunicarse a un conjunto heredado de datos orientado a recursos, mediante POJOs.
- Se desea transportar la capa de presentación para la creación de interfaces de usuario sobre la marcha.
- Las entidades JPA orientadas a objetos deben ser transformadas en un formato simplificado para su transferencia a través de RESTful, SOAP, CORBA, o incluso ASCII.

El DTO es una clase Java plana que implementa la interfaz Serializable, su estado es transportado como atributos, cada uno expuesto con accesores getter y setter. Pero su construcción también puede efectuarse directamente en el servicio, en un EntityManager, o ser encapsulada en un DAO (Bien, 2009).

La responsabilidad principal del Transfer Object es la optimización de la transferencia de datos.

### *Evolución del patrón*

En J2EE, el uso de un Transfer Object fue recomendado para ocultar detalles específicos de la persistencia gestionada por el contenedor, en la actualidad, las entidades JPA transfieren datos entre capas, especialmente en una única JVM. Los TOs son estructuralmente similares a las entidades JPA; por lo cual, cada cambio accionado por una solicitud llega a las entidades JPA y generalmente a los TOs.

### *Convenciones para el uso del patrón*

- La estructura de un TO es un POJO transferible que es creado directamente desde el almacén de datos o de la lógica del negocio.
- Los Objetos de Transferencia se construyen como JavaBeans compuestos de un constructor por defecto, y, getters y setters para cada atributo.
- Si la intención del uso de TOs es el desacoplamiento, deben ser almacenados en un subpaquete dedicado que pertenece a la transacción. Caso contrario, es suficiente colocarlos cerca del origen de su creación, por ejemplo, en el mismo paquete de los PDOs y DAOs.
- Se usa una convención de nombres como el sufijo TO para distinguirlos de los PDOs, por ejemplo, ClienteTO.
- Puede ser consumido y originado por la mayoría de los patrones y componentes. Los patrones Service o Service Facade, pueden devolver TOs para proporcionar vistas específicas del cliente a objetos del dominio.

```
public class ExternalizableBookDTO implements Externalizable{
private int numberOfPages;
private String name;
public ExternalizableBookDTO(int numberOfPages, String name) {
this.numberOfPages = numberOfPages;
this.name = name;
}
public void writeExternal(ObjectOutput out) throws IOException {
out.writeUTF(name);
out.writeInt(numberOfPages);
}
public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
this.name = in.readUTF();
this.numberOfPages = in.readInt();
}}
```

***Tabla 30: Ejemplo de implementación del patrón DTO***[95]

[95] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

Los patrones relacionados directamente con un DTO son Data Access Object, Service Facade y Service, estos patrones pueden consumir y producir DTOs además, entidades normales.

### *Estrategias de implementación*

La estrategia predeterminada para un TO es la implementación de una clase Java Serializable que se ajusta a un JavaBean. Tiene un constructor por defecto y cada atributo es expuesto a través de getters y setters. En la tabla 31 se da a conocer características de las estrategias existentes.

La utilización de este patrón afecta los siguientes puntos:

- No DRY: Los TOs deben mantenerse en sintonía con la entidad de negocio u otra representación del concepto.
- Abstracciones con fugas: Los DTOs se mantienen en paralelo a las entidades de dominio. De lo contrario los cambios en la lógica de negocio no serían visibles para los componentes de presentación.
- Semántica por valor: El nombre original de un TO fue value object, que involucraba semántica por valor. Un TO es una copia de un registro de la base de datos o una entidad JPA, por lo que cada DAO obtiene resultados de la operación en una nueva copia de un TO, en una sola transacción.

| Estrategia | Características |
|---|---|
| Builder-style TO | Un generador de estilo TO es un Objeto de Transferencia tradicional con una forma fluida de construcción.<br>El patrón Builder[96] es más conveniente para construir DTOs con numerosos atributos y permite la validación adicional del estado en tiempo de creación. |
| Builder-style Immutable TO | Con un patrón Builder se puede establecer el ajuste del atributo determinado.<br>Todos los atributos obligatorios son declarados como final y su existencia es verificada por el compilador.<br>Los parámetros obligatorios no tienen setters, deben pasar al builder del constructor explícitamente. |
| Client specific TO | Con el advenimiento de Java 6, enumeraciones y anotaciones, los datos y metadatos pueden ser transportados fácilmente con carga útil.<br>Los metadatos adicionales se dividen en dos categorías:<br>- Información de apoyo para la personalización o creación de la UI, y<br>-Metadatos de validación sintáctica de los atributos.<br>En los dos casos los metadatos son accesibles a través de la reflexión. |
| Generic DTO | El DTO Genérico permite el filtrado de atributos y la reducción bajo demanda.<br>Esta estrategia puede ser utilizada para la implementación de gestión de datos maestros o en aplicaciones genéricas como CRUD.<br>Su principal ventaja es también un inconveniente; la falta de tipificación hace su contenido muy dinámico, pero difícil de desarrollar y mantener. |

***Tabla 31: Características de las estrategias de implementación del patrón DTO***[97]

[96] **Builder (constructor virtual),** abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto (WWW34).

[97] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

*3.13.4.3.3 Legacy Pojo Integration.*

Existe la necesidad de integrar POJOs ya desarrollados a una aplicación Java EE, y acceder a ellos de forma sencilla y sin fricción. La Integración del POJO Heredado, debe cumplir las siguientes formalidades:

- La integración debería funcionar sin la modificación del código heredado, porque no siempre está disponible.
- La clase heredada debe ser capaz de aprovechar los servicios del contenedor y participar en las transacciones.
- El POJO heredado debe ser compatible con las restricciones de programación del EJB 3.
- El POJO heredado debe ser seguro para los subprocesos o ser capaz de desplegarse como un singleton.

El POJO heredado es desplegado como un EJB 3, para poder participar fácilmente en las transacciones y ser gestionado por el contenedor. También podría ser inyectado a un bean de sesión ya existente.

***Evolución del patrón***

En J2EE, no fue posible desplegar POJOs como EJB 2 sin cambiar el código. La infraestructura existente del POJO fue tratada como incompatible y tuvo que ser envuelta con un EJB 2.

Los objetos EJB 3 son POJOs anotados, por lo que la opción de desplegar POJOs existentes como componentes EJB 3 está disponible y es viable. La sobrecarga introducida por el contenedor EJB 3 es baja y las aplicaciones construidas con estos componentes son ligeras.

### *Convenciones del uso del patrón*

El despliegue del POJO heredado como un bean de sesión, se proporciona como un descriptor ejb-jar.xml. Los EJBs regulares pueden obtener acceso al POJO heredado a través de la Inyección de Dependencia. Este procedimiento es totalmente transparente para el resto de la aplicación, el POJO se convierte en un EJB sin modificar el código.

```
<ejb-jar>
<enterprise-beans>
<session>
<ejb-name>Table</ejb-name>
<business-local>javax.swing.table.TableModel</business-local>
<ejb-class>javax.swing.table.DefaultTableModel</ejb-class>
<session-type>Stateless</session-type>
</session>
</enterprise-beans>
</ejb-jar>
```

***Tabla 32: Ejemplo de implementación de Legacy Pojo Integration***[98]

El uso del patron Legacy Pojo Integration genera las siguientes consecuencias:

- Portabilidad: El código heredado no cambia, solo se proporcionan metadatos adicionales. Los POJOs heredados pueden ser usados en otro contexto.
- Rendimiento: La sobrecarga EJB 3 es muy baja, se obtendrá un rendimiento óptimo.
- Escalabilidad: Los EJBs pueden ser considerados neutrales respecto a la escalabilidad. La clusterización está disponible sólo a través de interfaces de negocio remotas y los POJOs heredados son accedidos siempre localmente.

[98] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

*3.13.4.3.4 Generic JCA.*

Un conector JCA puede acceder a cada recurso EJB incompatible, es portable, estandarizado, de fácil acceso y administrado por el servidor de aplicaciones. Debe satisfacer las siguientes exigencias:

- El desarrollador debe interactuar con los recursos incompatibles de Java EE.
- Se necesita acceso transaccional a los recursos heredados.
- Los recursos heredados no pueden escalar infinitamente por razones técnicas y de licenciamiento, se debe regular el rendimiento.
- El recurso heredado debe ser accesible desde EJBs.
- La solución debe ser portable a través de servidores.
- Se requiere administrar y monitorear el conector en tiempo de ejecución desde la consola de administración.

***Características del patrón***

- Es un conjunto de interfaces API y SPI orientado a la conexión.
- Se compone de dos interfaces, cuatro clases, y un archivo XML.
- Participa en las transacciones declarativas iniciadas por el EJB.
- Es completamente transparente para el desarrollador.

***Evolución del patrón***

El conector JCA no es nuevo, la versión 1.0 ha estado disponible desde el 2001 y fue una de las primeras especificaciones. La versión 1.5 está disponible desde el 2003 y está especificada en la JSR 112.

### *Convenciones de uso del patrón*

La implementación del adaptador JCA es una extensión del servidor de aplicaciones, no código de la aplicación. No obstante, es razonable seguir algunas convenciones:

- La implementación del conector debe estar ubicada en el paquete de integración.
- Si el desarrollador va a proporcionar una interfaz de conector personalizada, debe separar la implementación interna de la API pública.
- Independientemente de la complejidad de la implementación interna del conector, la API pública debe ser diseñada simple y convenientemente.

No existen estrategias notables para su implementación. El desarrollador puede determinar una manera óptima de implementación del conector JCA, dependiendo de sus necesidades. Su uso afecta los siguientes puntos:

- Portabilidad: La portabilidad de los adaptadores JCA no necesita ninguna extensión específica del proveedor.
- Escalabilidad: Como JCA es una especificación, la escalabilidad dependerá exclusivamente de su implementación y de la escalabilidad de los recursos heredados.
- Robustez: Es altamente dependiente de la implementación del conector, pero la disponibilidad de los contratos del sistema como la administración del ciclo de vida, transacción y conexión, permiten la realización de integraciones robustas.
- Complejidad: Una implementación parcial de un adaptador JCA puede ser más fácil y más rápida que un intento de construir una solución comparable desde cero.

- Consistencia: Los adaptadores JCA pueden participar en transacciones locales y distribuidas. El servidor de aplicaciones propaga las transacciones al adaptador de recursos y notifica su implementación sobre el estado actual de la transacción.
- Capacidad de mantenimiento: El adaptador JCA exige la reutilización. Un conector se despliega independientemente de las aplicaciones y evita la repetición de código. La separación estricta de la API pública y el SPI interno permite cambios en la implementación del conector interno sin romper sus clientes.

```
package com.abien.patterns.integration.genericjca;
import java.io.PrintWriter;
import javax.naming.Reference;
import javax.resource.spi.ConnectionManager;
import javax.resource.spi.ConnectionRequestInfo;
import javax.resource.spi.ManagedConnectionFactory;
public class FileDataSource  implements DataSource {
private ManagedConnectionFactory mcf;
private Reference reference;
private ConnectionManager cm;
private PrintWriter out;
public FileDataSource(PrintWriter out,ManagedConnectionFactory mcf,
ConnectionManager cm) {
out.println("#FileDataSource");
this.mcf = mcf;
this.cm = cm;
this.out = out;
} }
```

***Tabla 33: Ejemplo de implementación del patrón JCA genérico***[99]

La integración de servicios asíncrona puede ser considerada como una variante reducida del patrón JCA genérico. En la tabla 34, se indica la relación de este patrón con otros expuestos en este catálogo.

[99] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

| Patrones relacionados | Descripción |
|---|---|
| DAO, Service | Un adaptador de recursos JCA es utilizado por estos patrones. |
| Services, Service Facades | Los conectores de alto nivel pueden ser accedidos por estos patrones. |

***Tabla 34: Patrones relacionados con el patrón Generic JCA***[100]

*3.13.4.3.5 Asynchronous Resource Integrator (Service Activator).*

El patrón Asynchronous Resource Integrator es un Service Activator con un alcance y responsabilidades ampliadas, aunque su principal objetivo sigue siendo la integración de recursos incompatibles. La invocación asíncrona de servicios sincrónicos está cubierta por la especificación EJB 3.1. Debe solventar los siguientes requerimientos:

- Se necesita comunicación bidireccional entre Java EE y los servicios back-end.
- Se requiere acceso robusto a los recursos heredados.
- Se necesita una manera de acceder a los servicios asíncronos.
- Es necesaria una solución que integre trasparentemente los servicios existentes de Java EE con los servicios back-end.
- Es importante proporcionar un alto nivel de consistencia, escalabilidad, seguridad y robustez. Se requiere acceso transaccional.
- Se necesita intercambiar mensajes a través de protocolos como HTTP, REST o SOAP.

[100] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Evolución del patrón*

En EJB 2, un bean controlado por mensajes era la única manera de invocar EJBs síncronos en modo asíncrono. El Service Activator fue en realidad un patrón decorator o adapter[101], que utilizó de forma inadecuada el middleware orientado a mensajes, para invocar componentes sincrónicos de forma asincrónica. En EJB 3.1, solamente se debe anotar un método con @Asynchronous y no se necesita ninguna solución temporal.

Para la integración de recursos asíncronos en la capa de presentación con el patrón Asynchronous Resource Integrator, se deben configurar puentes HTTP o RESTful. Los recursos heredados, se basan en un protocolo binario ya existente.

### *Convenciones para el uso del patrón*

- El bean controlado por mensajes es responsable del reenvío del payload desde un mensaje al servicio, no debe contener lógica de negocio.
- Debe ser nombrado como el servicio con el sufijo ASI, abreviatura de integración de servicios asíncronos, o simplemente Listener.
- Puede residir en el mismo paquete del Servicio.
- El ASI tiene responsabilidades similares a un patrón Service Facade, por lo que puede ser colocado en el subpaquete de la fachada o frontera.
- Para la comunicación con los servicios back end, el bean controlado por mensajes pertenece a la capa de integración y debe estar en un subpaquete de esta capa.

[101] **Adapter (adaptador),** adapta una interfaz para que pueda ser manipulada por una clase que de otro forma no se podría utilizar (WWW34).

La implementación de JMS es importante en este patrón, el proveedor de mensajes es responsable de proporcionar la funcionalidad de adhesión y librerías de cliente, para Java y para las partes heredadas.

### *Estrategias de implementación*

Desde el punto de vista conceptual, existen dos estrategias disponibles cuya implementación es similar. La única diferencia es la fuente de los eventos entrantes. Se detallan en la tabla 35.

| Estrategia | Características |
|---|---|
| Front end Integration | Espera mensajes de la capa de presentación.<br>Los mensajes contienen la entrada del usuario como payload y tienen la granularidad de los parámetros del Service Facade.<br>Las transformaciones no son necesarias; el payload de los mensajes entrantes frecuentemente contiene objetos JSON o XML, que pueden ser consumidos directamente.<br>El bean controlado por mensajes extrae payload y la delega como un parámetro a un Servicio. |
| Back end Integration | Un sistema heredado es la fuente de los mensajes entrantes y, no tendrá gran influencia en la payload, tipo, codificación y contenido del mensaje.<br>Las transformaciones complejas no se deben implementar en el ASI, pero pueden ser factorizadas en un Servicio dedicado de conversión comprobable. |

***Tabla 35: Características de las estrategias de implementación del patrón Asynchronous Resource Integrator***[102]

[102] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

El uso del patrón Asynchronous Resource Integrator afecta a:

- Escalabilidad: Las transacciones son más cortas, el cliente no debe esperar hasta que la transacción termine para enviar otro mensaje.
- Complejidad: En caso de que el recurso heredado sea capaz de comunicarse con el servidor de mensajería, el ASI se convierte en la solución más simple y robusta.
- Acoplamiento débil: La dependencia del mensaje es débil y el productor puede ser cambiado y mantenido sin afectar al consumidor.
- Robustez: El desarrollador debe proporcionar robustez en tiempo de ejecución para alcanzar estabilidad.
- Portabilidad: La aplicación es dependiente únicamente de la API del JMS.
- Rendimiento: La comunicación bidireccional requiere al menos tres transacciones que afectan al servidor y por ende al rendimiento. Esta misma lógica de negocio puede ser invocada con una sola transacción en forma sincrónica.

El patrón Asynchronous Resource Integrator es activado por recursos back end o directamente por los clientes. La tabla 36, muestra su relación con otros patrones.

| Patrones relacionados | Descripción |
|---|---|
| Service | El patrón Asynchronous Resource Integrator reenvía payload sólo a los Servicios.<br>Los servicios pueden invocarlo mediante el envío de mensajes JMS. |
| DAOs | El Asynchronous Resource Integrator, en determinados casos, puede trabajar directamente con DAOs. |

***Tabla 36: Patrones relacionados con el patrón Asynchronous Resource Integrator***[103]

### *3.13.4.4 Patrones de Infraestructura y utilidades.*

Existen patrones difíciles de clasificar porque pueden ser utilizados en todas las capas, o no todos son relevantes en un contexto general y pueden ser omitidos en la descripción de la arquitectura. Los patrones de infraestructura y utilidades que se pueden emplear en el desarrollo de las aplicaciones son:

#### *3.13.4.4.1 Service Starter.*

El uso del patrón Service Starter debe satisfacer las siguientes necesidades:

- Se necesita una forma portable para iniciar los servicios con impaciencia.
- La solución debe proporcionar enlaces extensibles mediante los cuales, beans de sesión sin estado existentes pueden ser inicializados.

El patrón Service Starter actúa como un wrapper, para inicializar un bean de sesión se debe inyectar la referencia y llamar el método deseado; esto hará que el contenedor inyecte e inicialice el bean.

[103] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Características*

- El proceso es iniciado por un Servlet o un bean Singleton.
- La inicialización se realiza directamente en la configuración de arranque, o es delegada a los beans de sesión mediante la invocación de métodos ficticios.

```
@Stateless
public class ExistingServiceBean {
@PostConstruct
public void onInitialize(){
System.out.println("#" + this.getClass().getSimpleName() + " @PostConstruct");
}
public void pleaseInitialize(){
System.out.println("#" + this.getClass()… + " pleaseInitialize()");
}
public String someBusinessLogic(){
return "Hello from bean";
}}
```

***Tabla 37: Ejemplo de implementación del Service Starter***[104]

### *Evolución del patrón*

Debido a las mejoras presentadas por la tecnología EJB 3.1, es posible transformar los Servlets de inicio existentes en beans Singleton y desechar WARs triviales.

El patrón Service Starter mejora la robustez del sistema en tiempo de ejecución, empezando servicios críticos con impaciencia. La tabla 38, indica la relación de este patrón con otros del catálogo expuesto.

[104] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

| Patrones relacionados | Descripción |
|---|---|
| Service, Service Facades, Gateways | El patrón Service Starter invoca estos patrones para producir su inicialización. |
| Singleton | Este patrón es técnicamente idéntico al Service Starter. |

***Tabla 38: Patrones relacionados con el patrón Service Starter***[105]

*3.13.4.4.2 Singleton.*

El patrón Singleton debe cumplir las siguientes exigencias:

- La solución debe ser portable a través de servidores.
- Un Singleton es compartido a través de peticiones de usuario. Es accedido en modo de sólo lectura o el contenedor provee un mecanismo de sincronización.
- No debe influir en la recolección de basura y no debe construirse con modificadores estáticos y sincronizados.

El patrón Singleton fue estandarizado con la especificación EJB 3.1. Tiene las siguientes características:

- Es un bean que existe una sola vez en una JVM y puede ser iniciado en tiempo de arranque del contenedor.
- En un entorno clusterizado, se tendrán múltiples instancias Singleton.
- Desde un punto de vista técnico, un Singleton es parecido al Service Starter, pero su responsabilidad principal no es la delegación de sus capacidades de arranque, sino proporcionar estado compartido y administración de concurrencia.

[105] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

- El bean Singleton proporciona un valor añadido ya implementado, por ejemplo, almacenamiento en caché o servicios de configuración.
- Puede acceder a Service Facades existentes, servicios, y recursos para buscar información necesaria y almacenarla en caché posteriormente.

```
@Singleton
public class MasterDataCache {
private Map<String,Object> cache;
@PostConstruct
public void initCache(){
this.cache = new HashMap<String, Object>();
}
public Object get(String key){
return this.cache.get(key);
}
public void store(String key,Object value){
this.cache.put(key, value);
}}
```

***Tabla 39: Ejemplo de implementación del patrón Singleton***[106]

### *Evolución del patrón*

El uso de campos estáticos está restringido. Como una solución alternativa, los conectores MBeans o JCA pueden ser utilizados. El acceso a JMX desde beans de sesión no se especifica y es propenso a errores. Los beans Singleton son simples, concisos, potentes y totalmente independientes de una implementación concreta del servidor de aplicaciones.

### *Estrategias de implementación*

La tabla 40, muestra las características de las estrategias que se pueden utilizar para implementar el patrón Singleton en una aplicación.

[106] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

| Estrategia | Características |
|---|---|
| Gatekeeper | Un Singleton puede ser utilizado para limitar la concurrencia a los recursos del servidor.<br>Puede proporcionar acceso secuencial a los métodos de los servicios heredados.<br>En un clúster se obtendrá una instancia del Singleton por nodo, y varias instancias por cada aplicación. |
| Caching Singleton | Los Singletons son habitualmente usados para envolver o implementar soluciones de almacenamiento en caché.<br>Esta estrategia es responsable del almacenamiento en caché y no de la recuperación de datos; pero, un Service inyectado podría obtener los datos si falla la caché, y, proporciona una fachada de almacenamiento.<br>Si la caché no se ha configurado correctamente, el servidor de aplicaciones lanzará una excepción y la aplicación no estará totalmente disponible. |

***Tabla 40: Características de las estrategias de implementación del patrón Singleton***[107]

El uso del patrón Singleton puede ingerir en:

- Escalabilidad: Un Singleton existe solo una vez en la JVM, por lo tanto es un cuello de botella natural, que afecta la capacidad de ampliación. La estrategia caching singleton puede guardar los accesos a la base de datos y mejorar la escalabilidad.
- Rendimiento: El Caching Singleton consigue mejorar el rendimiento, ya que el objeto requerido puede ser capturado directamente desde la memoria.

[107] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

- Robustez: La instancia del Singleton permanecerá en la memoria hasta el redespliegue o cierre de la aplicación. Por lo cual, es importante configurar la memoria caché apropiadamente y establecer la cantidad máxima de objetos a un valor razonable.

La estrategia Gatekeeper accede a recursos back-end y a servicios de integración. La relación del patrón Singleton con otros patrones, se ve reflejada en la tabla 41.

| Patrones relacionados | Descripción |
|---|---|
| Services, Service Facades, | Un Caching Singleton puede ser accedido por estos patrones. |
| Service Starter | Un Singleton está estrechamente relacionado con este patrón |

***Tabla 41: Patrones relacionados con el patrón Singleton***[108]

*3.13.4.4.3 Bean Locator.*

El patrón Bean Locator es responsable de localizar EJBs y de la construcción de los nombres JNDI globales. Debe satisfacer las siguientes necesidades:

- Los detalles de la API JNDI deben estar ocultos. El usuario debe ser independiente de esta API.
- El Bean Locator debe estar en capacidad de ser utilizado dentro o fuera del contenedor EJB.

[108] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

```
public class BeanLocator {
public static Object lookup(String jndiName) {
Context context = null;
try {
context = new InitialContext();
return context.lookup(jndiName);
} catch (NamingException ex) {
throw new IllegalStateException("...");
} finally {
try {
context.close();
}}
```

***Tabla 42: Ejemplo de implementación del Bean Locator***[109]

### *Evolución del patrón*

El Bean Locator puede ser considerado como una forma específica del patrón Service Locator de J2EE, que fue diseñado para buscar todo tipo de recursos JNDI y obtener interfaces de inicio. En Java EE, el patrón Bean Locator retorna las interfaces de negocio o instancias del bean, y, sólo debe utilizarse en clases donde la Inyección de Dependencia no funciona o no está disponible.

### *Características*

- El Bean Locator es utilizado por las clases cuando la Inyección de Dependencia no se encuentra disponible.
- Delega las llamadas al JNDI.
- Retorna beans remotos o locales con estado, sin estado y Singleton.
- Es usado en el momento de la creación de los componentes, producida en el despliegue de la aplicación.

[109] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

- Las anotaciones, beans sin estado y Singleton, podrían ser fácilmente almacenados en caché dentro del Bean Locator para mejorar el rendimiento.

***Estrategias de implementación***

El Bean Locator es una herramienta útil que puede ser extendida en muchas formas. Su utilidad puede ser personaliza para adaptarse a las necesidades de la aplicación.

El uso del patrón Bean Locator afecta los siguientes puntos:

- Portabilidad: El Bean Locator se basa en nombres JNDI globales. Es portable a través de servidores.
- Rendimiento y escalabilidad: El uso del Bean Locator puede afectar el tiempo de arranque, no debe haber ningún impacto en el rendimiento en tiempo de ejecución.
- Robustez: Cuando el Bean Locator se utiliza con impaciencia, no tiene ningún impacto negativo en la robustez. Es comparable con el mecanismo de Inyección de Dependencia habitual.

La Inyección de Dependencia puede ser considerada como un Bean Locator 2.0, porque utiliza una versión genérica de este patrón, configurada con metadatos derivados de convenciones, archivos de clase, anotaciones y XML. El bean localizado es inyectado automáticamente por el framework.

El patrón Service Locator de J2EE está relacionado con el Bean Locator; su implementación fue limitada debido a la falta de nombres JNDI globales y anotaciones.

*3.13.4.4.4 Thread Tracker.*

El uso de esta utilidad, debe solventar las siguientes exigencias:

- La solución debe ser portable a través de servidores.
- La ampliación de las capacidades de control debe estar claramente separada del código de negocio.
- El seguimiento del subproceso debe ser fácilmente activado y desactivado.
- La funcionalidad de control adicional debe ser fácil de remover antes del despliegue en producción.
- El seguimiento del subproceso debe ser genérico para ser aplicado a los beans ya existentes.

```
public class ThreadTracker {
@AroundInvoke
public Object annotateThread(InvocationContext invCtx) throws Exception{
String originName = Thread.currentThread().getName();
String beanName = invCtx.getTarget().getClass().getName();
String tracingName = beanName + "#" + invCtx.getMethod().getName() + " " +
originName;
Thread.currentThread().setName(originName);
} }
```

***Tabla 43: Ejemplo de implementación del Thread Tracker***[110]

***Evolución***

Con la llegada de EJB 3 y la introducción de los interceptores, las preocupaciones transversales pueden ser fácilmente factorizadas desde la lógica de negocio; esto se puede lograr sin frameworks o librerías.

[110] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Características*

- El Thread Tracker se puede conectar a una clase determinada empleando anotaciones o XML.
- Se puede aplicar a beans de sesión y beans controlados por mensajes.
- Si es aplicado en un Service Facade, el nombre del subproceso cambia al del método del Service Facade en ejecución y puede fácilmente controlar su rendimiento, desempeño, y escribir los registros de transacciones específicas.
- Es útil para la administración de beans controlados por mensajes.
- Se ejecuta de forma asincrónica en una transacción separada y un subproceso, que puede proceder de un pool distinto.
- Ayuda a encontrar potenciales cuellos de botella y métodos lentos.
- Para un control más ágil y la búsqueda de puntos de acceso, se podría desplegar Services con ThreadTracker.

El uso del Thread Tracker, ocasionará las siguientes secuelas:

- Portabilidad: El ThreadTracker no puede ser considerado como una solución portable. Su implementación es adecuada para un solo proyecto, pero debe ser evaluado para un entorno multi-servidor.
- Rendimiento: Cada direccionamiento indirecto disminuye el rendimiento. Aunque la sobrecarga del algoritmo es insignificante, la interceptación es varias veces más lenta, en comparación con una llamada local directa.

*3.13.4.4.5 Dependency Injection Extender.*

El Extensor de Inyección de Dependencia (DIE), debe cumplir los siguientes requisitos:

- Se necesita desplegar un componente existente del framework sin modificar la aplicación EJB.
- El componente de DI heredado debe intervenir en las transacciones EJB y en el contexto de seguridad.
- La integración debe ser transparente para el componente heredado.
- Los EJBs deben coexistir con clases administradas desde el framework integrado.
- Las clases administradas desde los frameworks deben ser inyectadas directamente en los EJBs.
- Es una exigencia declarar y asociar el interceptor con un bean, utilizando anotaciones o XML.

```
@Stateless
@Local(ServiceFacade.class)
@Interceptors(PerMethodGuiceInjector.class)
public class ServiceFacadeBean implements ServiceFacade{
@Inject
private MessageProvider message;
public String getHello(String msg){
return msg + " " + message.getMessage();
}}
```

***Tabla 44: Ejemplo de implementación del DIE***[111]

[111] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Evolución del patrón*

El patrón ofrece alternativas para la coexistencia de diferentes frameworks detrás de un Service Facade. La integración de los componentes heredados con EJBs no requiere ninguna configuración XML. Esto hace que este patrón, sea práctico para la reutilización de fragmentos pequeños de la funcionalidad existente.

### *Características*

- El interceptor desempeña el papel más importante; es el puente entre EJBs y componentes heredados.
- Es reutilizable a través de los servicios y Service Facades.
- Es responsable de:
    - La configuración del framework de DI.
    - La secuencia de arranque del framework DI heredado.
    - La inyección por instancia o por método de los componentes en el servicio o Service Facade.
- El bean de sesión únicamente debe exponer las dependencias a las clases administradas utilizando la semántica del framework heredado y declarando el interceptor.

### *Estrategias de implementación*

En la tabla 45, se detallan ciertas estrategias que pueden ser útiles en la implementación del patrón.

| Estrategia | Características |
|---|---|
| Stateful Session Bean Injector | Los componentes inyectados pueden almacenarse en caché durante la duración de la sesión.<br>El framework heredado, también se convierte en responsable de la creación de componentes con estado en el momento de la inyección. |
| Stateless Session Bean Injector | Los componentes deben ser inyectados antes de cada llamada a un método o pueden usar proxies inteligentes que reaccionan.<br>Los componentes idempotentes pueden ser inyectados una vez y reutilizados entre instancias del bean de sesión. |
| Transactional Integration | Los recursos transaccionales son dados de alta en la transacción automáticamente por el contenedor EJB.<br>Las clases administradas sólo buscan o inyectan recursos transaccionales desde el contexto JNDI y lo usan.<br>Se puede también inyectar el mismo recurso en un bean de sesión habitual sin perder la consistencia. |

***Tabla 45: Características de las estrategias de implementación del patrón Dependency Injection Extender***[112]

El uso del patrón Dependency Injection Extender, acarrea las siguientes consecuencias:

- Rendimiento: Los interceptores y la DI se basan en la reflexión. Pero el proceso de inyección es mucho más rápido que el acceso normal a la base de datos.
- Portabilidad: El DIE se apoya en interceptores y no sólo en utilizar las funciones de los servidores de aplicaciones propietarios. Es portable a través de servidores de aplicaciones de Java EE.

[112] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

Los patrones basados en interceptores están relacionados al Extender Dependency Injection, especialmente la estrategia SOA del Service Facade y el Payload Extractor.

El DIE se aplica con frecuencia en Service Facades o, en componentes más grandes sobre los servicios, para exponer el framework heredado en forma de EJB a la capa de presentación. El DAO también podría depender de los recursos propietarios e integrar componentes del framewok.

*3.13.4.4.6 Payload Estractor.*

El uso del Payload Estractor debe solventar los siguientes requisitos:

- Los mensajes envenenados deben ser manejados adecuadamente. Deben ser redirigidos a una cola de mensajes específica y persistente.
- La comprobación de tipos y gestión de errores son reutilizables y deben ser factorizadas fuera de los beans dirigidos por mensajes en un componente dedicado.
- El manejo de errores debe ser configurado e implementado fuera de la lógica de negocio.
- La solución debe ser transaccional.

El interceptor del Payload Extractor se invoca antes del método, recibe el mensaje real y detecta su tipo. Dependiendo del tipo, lanza el mensaje, extrae su contenido, e invoca el método de consumo usando reflexión.

```
public class PayloadExtractor {
public static final String CONSUME_METHOD = "consume";
@EJB
private DeadLetterHandler dlh;
@AroundInvoke
public Object audit(InvocationContext invocationContext) throws Exception{
Object mdb = invocationContext.getTarget();
Method method = invocationContext.getMethod();
if("onMessage".equals(method.getName())){
Message jmsMessage = (Message) messageParameter(invocationContext);
invokeMethodWithSingleParameter(jmsMessage,mdb);
} }
```

***Tabla 46: Ejemplo de implementación del Payload Estractor***[113]

### *Evolución del patrón*

En J2EE 1.4 fue imposible decorar beans controlados por mensajes sin la ampliación del código byte. El desarrollador debía desplazarse en una superclase abstracta, lo cual fue intrusivo y complejo. Los interceptores introdujeron una alternativa más pragmática y ligera. “Con el patrón Payload Extractor se puede iniciar inmediatamente el comportamiento predeterminado e implementar código personalizado bajo demanda” (Bien, 2011).

### *Características*

El interceptor Payload Extractor es la parte clave de este patrón. Es responsable de:

- La intercepción del método del mensaje y el secuestro de su parámetro.
- Emitir el mensaje en una interfaz concreta.
- Extracción de payload.

[113] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

- La invocación de un método del negocio con el payload como parámetro.
- Escalada adecuada de errores irrecuperables invocando el bean controlador de mensajes.

La responsabilidad restante del bean gestionado por mensajes es sólo la delegación del payload extraído a un servicio.

La utilización del Payload Extractor afecta los siguientes puntos:

- Rendimiento: Si se está empleando un protocolo distribuido o XML como payload, tardará pocos milisegundos en consumir y procesar el mensaje. En sistemas de alto rendimiento, la interceptación podría convertirse en un problema.
- Mantenimiento: Simplifica y centraliza el código de manejo de errores como una preocupación transversal. No es intrusivo, se puede fácilmente desactivar la interceptación y proporcionar una propia lógica de extracción.
- Complejidad: La extracción es recurrente y la lógica de manejo de errores se implementa una vez y se factoriza en el interceptor como una preocupación transversal. Los beans dirigidos por mensajes están casi vacíos, lo cual reduce la complejidad del código.

Desde el punto de vista de la implementación, el patrón Payload Extractor es similar al Dependency Injection Extender. El Payload Extractor reenvía el contenido del mensaje a los servicios, que son inyectados en el bean controlado por mensajes.

*3.13.4.4.7 Resource Binder.*

La Carpeta de Recursos debe satisfacer las siguientes necesidades:

- Se necesita una forma conveniente y robusta para registrar recursos personalizados.
- El registro debe ser independiente del servidor de aplicaciones.
- Los recursos deben ser inyectables en los EJBs.
- No se requiere una complicada administración del ciclo de vida: los recursos no tienen estado y pueden ser conservados como un patrón Singleton, o un Flyweight[114].

```
public class CustomResource{
public final static String JNDI_NAME = "theName";
public void doSomething(){
System.out.println("#Done !");
}}
```

***Tabla 47: Ejemplo de implementación del Resource Binder***[115]

### *Evolución del patrón*

El patrón Singleton de Java EE, permite registrar los recursos personalizados antes de la inicialización de la lógica de negocio. Antes de EJB 3.1, la inicialización era posible solamente en un Servlet o usando clases propietarias de inicio. La inyección de dependencia hace del patrón Resource Binder una opción agradable. El uso de JNDI sin DI es muy minucioso, propenso a errores, y complejo.

[114] **Flyweight (peso ligero),** reduce la redundancia cuando gran cantidad de objetos poseen idéntica información (WWW34).

[115] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Características*

El patrón Resource Binder es responsable de:

- La inicialización de los recursos necesarios antes de cualquier invocación de la lógica de negocio.
- Administración del ciclo de vida del recurso personalizado.
- Registro del recurso en el árbol JNDI del servidor de aplicaciones.
- Recurso de limpieza y cancelación del registro en el momento de apagar el servidor.

### *Estrategias de implementación*

Las implementaciones JNDI propietarias, pueden requerir medidas específicas para enlazar un POJO al árbol JNDI. El desarrollador puede registrar un adaptador que implemente las interfaces necesarias y sostenga el recurso; es decir, el adaptador será inyectado.

Si los recursos son reutilizables en diferentes aplicaciones, puede empaquetar el Resource Binder con todos los recursos en un archivo específico. La implementación del patrón podría conservarse separadamente, pero todavía se necesitaría tener los recursos personalizados en el classpath.

El uso de Resource Binder afecta los siguientes puntos:

- Portabilidad: JNDI es parte del JDK, pero no se debe asumir que el Resource Binder trabajará con cada servidor de aplicaciones fuera del contenedor.

- Mantenimiento: La inyección de recursos personalizados reduce la cantidad de código de infraestructura. No hay que conservar el manejo de excepciones de los métodos de búsqueda o service locators.
- Rendimiento: El Resource Binder no tiene ningún efecto sobre el rendimiento en tiempo de ejecución, sin embargo, puede afectar el tiempo de inicio. El registro del recurso y la inyección, son realizados antes de cualquier llamada de negocio.
- Escalabilidad: No afecta la capacidad de ampliación; sin embargo, es un Singleton y los POJOs son registrados globalmente al iniciarse la aplicación. La escalabilidad de la solución depende de la escalabilidad de los recursos inyectados.

En la tabla 48, se indican los patrones relacionados al Resource Binder.

| Patrones relacionados | Descripción |
|---|---|
| Service Locator | Este patrón está inversamente relacionado con el Resource Binder, ya que es responsable de buscar activamente los recursos del árbol JNDI, mientras que el Resource Binder permite la Inyección de Dependencia de los recursos personalizados. |
| Services, Service Facades, DAOs. | Los recursos expuestos por el patrón Resource Binder son consumidos en beans de sesión y por lo tanto en estos patrones. |

***Tabla 48: Patrones relacionados con el patrón Resource Binder***[116]

[116] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

*3.13.4.4.8 Context Holder.*

El patrón Soporte de Contexto debe satisfacer las siguientes necesidades:

- Es necesario adjuntar objetos adicionales al subproceso actual sin violar las restricciones de programación.
- El servidor de aplicaciones puede usar varios pools de subprocesos dedicados, por lo que la información almacenada en el subproceso local puede perderse.
- Se necesita una solución portable.

```
import static kitchensink.contextholder.RegistryKey.*;
public class CurrentTimeMillisProvider {
@Resource
private TransactionSynchronizationRegistry registry;
@AroundInvoke
public Object injectMap(InvocationContext ic) throws Exception{
registry.putResource(KEY, System.currentTimeMillis());
return ic.proceed();
}}
```

***Tabla 49: Ejemplo de implementación del patrón Context Holder***[117]

### *Evolución del patrón*

El Context Holder estaba disponible en J2EE 1.4, pero debía ser recuperado mediante el mecanismo de búsqueda JNDI con un nombre estandarizado; lo cual requiere varias líneas de código con un manejo de excepciones apropiado o el uso de un Service Locator. Java EE 5 introdujo Inyección de Dependencia, lo que simplifica el acceso a los recursos JNDI.

[117] Ejemplo tomado del libro (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

### *Características*

- El Context Holder es proporcionado por el contenedor o por Java SE.
- En los dos casos, es iniciado antes de la primera invocación del método del negocio de un Service Facade o Gateway. Esto se puede realizar con un interceptor, proxy dinámico, o filtro Servlet que es el inyector.
- El inyector extrae o crea información contextual y lo almacena en el context holder.
- Los participantes restantes tienen solamente que buscar este patrón para acceder a la información.
- Un inyector explícito es opcional ya que el context holder, puede ser inyectado en la lógica de negocio y ser usado como un transporte conveniente de datos técnicos y transversales.

### *Estrategias de implementación*

En la tabla 50, se exponen las estrategias que pueden ser empleadas para la implementación del patrón Context Holder en una aplicación. Su uso afecta a:

- Portabilidad: La solución basada en TransactionSynchronizationRegistry está respaldada por la especificación Java EE 6 y debe ser portable a través de servidores de aplicaciones. La estrategia basada en ThreadLocal funciona cuando la invocación completa se ejecuta en un solo subproceso de ejecución.
- Rendimiento: El rendimiento de la estrategia ThreadLocal es altamente dependiente de la versión de Java SE. El rendimiento de la solución basada en TransactionSynchronizationRegistry es dependiente de la implementación de las aplicaciones.

- Capacidad de mantenimiento: El Context Holder reduce la cantidad de código necesario para el transporte de datos contextuales transversales. La solicitud puede ser enriquecida con información adicional sin cambiar la infraestructura existente.
- Flexibilidad: La estrategia Thread Local no necesita transacciones o APIs JTA para trasportar datos entre objetos y también puede ser usada en un contenedor web simple. Sin embargo, limita toda la invocación a una relación 1:1 entre la solicitud y el subproceso. La estrategia TransactionSynchronizationRegistry no tiene esta limitación, pero depende de la existencia de transacciones.

| **Estrategia** | **Características** |
|---|---|
| Estrategia TransactionSynchronizationRegistry | Es portable a través de servidores de aplicaciones e independiente de la configuración individual, y propietaria del pool de subprocesos.<br>La transacción actual se utiliza como mecanismo de transporte para el almacenamiento de datos adicionales. |
| Estrategia ThreadLocal | Es menos portable.<br>Funciona muy bien si la invocación completa es ejecutada en un solo subproceso.<br>No requiere Inyección de Dependencia.<br>Puede ser usada en un contenedor web para el transporte de datos. |

***Tabla 50: Características de las estrategias de implementación del patrón Context Holder***[118]

[118] Información obtenida del libro (Real World Java EE Patterns – Rethinking Best Practices, 2009).

El Context Holder es utilizado por el patrón Service Facade y sus interceptores para el transporte de datos a los servicios y DAOs. Debido a que el Context Holder utiliza interceptores, este patrón es similar a los patrones Thread Tracker, Payload Extractor, e Dependency Injection Extender.

# CAPÍTULO IV

# 4 Aplicativo

## 4.1 Introducción

En la sección ***Gestión del Proyecto*** se indican las planificaciones del desarrollo del proyecto, así como el cronograma de ejecución del proyecto, de construcción de la aplicación y cumplimiento de los plazos estimados.

En la sección ***Modelado del Negocio*** se encuentran los artefactos de la metodología RUP utilizados para definir un modelo del negocio, modelos de objetos del negocio y el modelo del dominio.

En la sección ***Requisitos*** se detallan los artefactos definidos según la metodología RUP, es decir, los documentos: plan de desarrollo, visión, glosario, matrices de atributos de los requerimientos, los casos de uso y sus especificaciones.

En la sección ***Análisis/Diseño*** se expone tanto el modelo de datos (modelo entidad - relación).

En la sección ***Implementación*** se visualizan los prototipos de las interfaces de usuario de la aplicación.

Por último, en el apartado ***Pruebas***, se encuentran las especificaciones de casos de pruebas funcionales.

A continuación se indican las herramientas y tecnologías que fueron utilizadas en el desarrollo del sistema:

**ENTORNO DE DESARROLLO:**

**Hardware:**

**Marca:** Sony Vaio
**Modelo:** VGN – CS120J
**Procesador:** Intel Core 2 Duo 2.26GHz
**Memoria RAM:** 4GB
**Disco Duro:** 320 GB

**Software:**

**Sistema Operativo:** Microsoft Windows Vista Home Premium 64 bits
**Lenguaje de programación:** Java
**Sistema de Gestión de Base de Datos Relacional:** PostgreSql 9.0
**Entorno de desarrollo:** NetBeans 7.2.1
**Servidor de aplicaciones:** GlassFish 3.1
**Framework:** JSF 2.0
**Modelador UML:** ArgoUML 0.28.1

## 4.2 Gestión del Proyecto

En este apartado se detalla la planificación inicial del proyecto para las fases de inicio y de elaboración (según la definición de la metodología RUP).

### 4.2.1 Plan de Desarrollo de Software.

#### *4.2.1.1 Introducción.*

Este Plan de Desarrollo de Software es una versión preliminar preparada para ser incluida en la propuesta elaborada como respuesta al trabajo final de grado previo a la obtención del título de Ingeniero en Sistemas Computacionales de la Facultad de Ingeniería en Ciencias Aplicadas de la Universidad Técnica del Norte.

El enfoque de desarrollo propuesto constituye una configuración del proceso RUP de acuerdo a las características del proyecto, seleccionando los roles de los participantes, las actividades a realizar y los artefactos (entregables) que serán generados. Este documento es a la vez uno de los artefactos de RUP.

##### *4.2.1.1.1 Propósito.*

El propósito del Plan de Desarrollo de Software es proporcionar la información necesaria para controlar el proyecto. En él se describe el enfoque del desarrollo de software.

Los usuarios del Plan de Desarrollo de Software son:

- El jefe de proyecto, quien utiliza este documento para organizar la agenda y necesidades de recursos, y para realizar su seguimiento.
- Los miembros del equipo de desarrollo, lo usan para entender lo que deben hacer, cuándo deben hacerlo y qué otras actividades dependen de ello.

*4.2.1.1.2 Alcance.*

El Plan de Desarrollo de Software describe el plan global usado para el desarrollo del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro.

Durante el proceso de desarrollo, en el artefacto “Visión”, se definen las características del producto a desarrollar, lo cual constituye la base para la planificación de las iteraciones.

Para la versión 1.0 del Plan de Desarrollo de Software se ha hecho un enfoque en la captura de requisitos por medio del stakeholder representante de la empresa para hacer una estimación aproximada, una vez emprendido el proyecto y durante la fase de Inicio se generará la primera versión del artefacto Visión, el cual será utilizado para refinar este documento.

Posteriormente, el avance del proyecto y el seguimiento en cada una de las iteraciones ocasionará el ajuste de este documento produciendo nuevas versiones actualizadas.

*4.2.1.1.3 Resumen.*

Después de esta introducción, el documento restante está organizado en las siguientes secciones:

***Vista General del Proyecto.-*** Proporciona una descripción del propósito, alcance y objetivos del proyecto, estableciendo los artefactos que serán producidos y utilizados durante el mismo.

***Organización del proyecto.-*** Describe la estructura organizacional del equipo de desarrollo.

***Gestión del proceso.-*** Explica los costos y planificación estimada, define las fases e hitos del proyecto y describe cómo se realizará su seguimiento.

***Planes y guías de aplicación.-*** Proporciona una vista global del proceso de desarrollo de software, incluyendo métodos, herramientas y técnicas que serán utilizadas.

### *4.2.1.2 Vista general del proyecto.*

*4.2.1.2.1 Propósito, alcance y objetivos.*

La información que a continuación se incluye ha sido extraída de las diferentes reuniones que se han efectuado con el stakeholder de la empresa desde el inicio del proyecto.

El proyecto debe proporcionar una respuesta para el desarrollo de los módulos implicados en el sistema hotelero para Santa Agua de Chachimbiro. Estos módulos son los siguientes:

- Módulo Reservación:
  - Control de disponibilidad de cabañas.
  - Introducción de datos del cliente.
  - Consultas de reservaciones.
  - Dar de alta y cancelar reservas.
  - Ofrecer el servicio de pagos on-line.

- Módulo Recepción:
  - Realizar el registro de huéspedes con o sin reservación.
  - Modificación de datos de reserva y recepción.
  - Consulta de disponibilidad cabañas.
  - Consultas de reservaciones.

- Módulo Facturación
  - Emisión de facturas.

- Módulo Seguridad
  - Creación y eliminación de usuarios.
  - Gestión de acceso al sistema.

- Módulo Reportes
  - Listado de reservaciones.
  - Llegadas y salidas.
  - Ocupación de habitaciones.
  - Total de ingresos.

Para el servicio de pagos on-line se plantea el uso de PayPal por ser el sistema de pagos online más utilizado del mercado.

*4.2.1.2.2 Suposiciones y restricciones.*

Las suposiciones y restricciones del sistema, derivadas directamente de las entrevistas con el stakeholder de la empresa son:

a) Deben considerarse las implicaciones de los siguientes puntos críticos:
    a. Sistemas seguros: protección de la información y seguridad en las transmisiones de datos.
    b. Gestión de flujos de trabajo, seguridad de las transacciones e intercambio de información.

b) No se puede realizar la reserva total del complejo. Solamente si el cliente realizara la reservación total de las cabañas, cuya capacidad es de 130 personas, se incluirá la alimentación.

Como es natural, la lista de suposiciones y restricciones se incrementará durante el desarrollo del proyecto, particularmente una vez establecido el artefacto “Visión”.

*4.2.1.2.3 Entregables del proyecto.*

A continuación se indican y describen cada uno de los artefactos que serán generados y utilizados por el proyecto y que constituyen los entregables. Esta lista constituye la configuración de RUP desde la perspectiva de artefactos, y que es propuesta para este proyecto.

Es preciso enfatizar que de acuerdo a la filosofía RUP (y de todo proceso iterativo e incremental), todos los artefactos son objeto de modificaciones a lo largo del proceso de desarrollo, con lo cual, solo al término del proceso se podrá tener una versión definitiva y completa de cada uno de ellos.

Sin embargo, el resultado de cada iteración y los hitos del proyecto están enfocados en conseguir un cierto grado de completitud y estabilidad de los artefactos. Esto será indicado posteriormente, cuando se presenten objetivos de cada iteración.

**Plan de Desarrollo de Software**

Es el presente documento.

**Modelos de Casos de Uso del Negocio**

Es un modelo de las funciones de negocio vistas desde la perspectiva de los actores externos (agentes de registro, solicitantes finales, otros sistemas, etc.), permite situar el sistema en el contexto organizacional haciendo énfasis en los objetivos en este ámbito. Este modelo se representa con un Diagrama de Casos de Uso usando estereotipos específicos para este modelo.

**Glosario**

Es un documento que define los principales términos usados en el proyecto.

**Modelo de Casos de Uso**

El Modelo de Casos de Uso presenta las funciones del sistema y los actores que hacen uso de ellas. Se representa a través de Diagramas de Casos de Uso.

**Visión**

Este documento define la visión del producto desde la perspectiva del cliente, especificando las necesidades y características del producto. Constituye una base de acuerdo a los requerimientos del sistema.

**Especificaciones de Casos de Uso**

Para los casos de uso que lo necesiten (cuya funcionalidad no sea evidente o que no baste con una simple descripción narrativa) se realiza una descripción detallada utilizando una plantilla de documento, donde se incluyen: precondiciones, post – condiciones, flujos de eventos y requisitos no funcionales asociados. También para casos de uso cuyo flujo de eventos sea complejo, podrá adjuntarse una representación gráfica mediante un Diagrama de Actividad.

**Prototipos de Interfaces de Usuario**

Se trata de prototipos que permiten al usuario hacerse una idea casi precisa de las interfaces que proveerá el sistema y así, conseguir una retroalimentación de su parte respecto a los requisitos del sistema. Estos prototipos se realizarán como: dibujos a mano en papel, dibujos con alguna herramienta gráfica o prototipos ejecutables interactivos, siguiendo ese orden de acuerdo al avance del proyecto.

**Modelo de Análisis y Diseño**

Este modelo establece la realización de casos de uso en clases y pasando desde una representación en términos de análisis (sin incluir aspectos de implementación) hacia una de diseño (incluyendo una orientación hacia el entorno de implementación), de acuerdo al avance del proyecto.

**Casos de Prueba**

Cada prueba es especificada mediante un documento que establece las condiciones de ejecución, las entradas de la prueba, y los resultados esperados. Estos casos de prueba son aplicados como pruebas de regresión en cada iteración. Cada caso de prueba llevará asociado un procedimiento de prueba con las instrucciones para realizar la prueba.

**Plan de Iteración**

Es un conjunto de actividades y tareas ordenadas temporalmente, con recursos asignados y dependencias entre ellas.

**Lista de Riesgos**

Este documento incluye una lista de riesgos conocidos y vigentes en el proyecto, ordenados en orden decreciente de importancia y con acciones específicas de contingencia o para su mitigación.

**Manual de Instalación**

Este documento incluye las instrucciones para realizar la instalación del producto.

**Materia de Apoyo al Usuario Final**

Corresponde a un conjunto de documentos y facilidades de uso del sistema, incluyendo: Guías del Usuario, Guías de Operación, Guías de Mantenimiento y Sistema de Ayuda en línea.

*4.2.1.2.4 Evolución del plan de desarrollo de software.*

El Plan de Desarrollo de Software se revisará y refinará al comienzo de cada iteración.

#### *4.2.1.3 Organización del proyecto.*

##### *4.2.1.3.1 Participantes en el proyecto.*

El personal participante en el proyecto, considerando las fases de Inicio, Elaboración, Construcción, estará formado por los siguientes puestos de trabajo y personal asociado:

**Jefe de Proyecto:** Ing. José Luis Rodríguez.
**Arquitecto de Software:** Egda. Maricruz Acosta Yerovi.
**Ingeniero de Software:** Egda. Maricruz Acosta Yerovi.
**Programador:** Egda. Maricruz Acosta Yerovi.

##### *4.2.1.3.2 Interfaces externas.*

La empresa definirá los participantes del proyecto que proporcionarán los requisitos del sistema, y entre ellos, quienes serán los encargados de evaluar los artefactos de acuerdo a cada módulo y según el plan establecido.

El equipo de desarrollo interactuará activamente con los participantes de la empresa para la especificación y validación de los artefactos generados.

##### *4.2.1.3.3 Roles y responsabilidades.*

A continuación se describen las principales responsabilidades de cada uno de los puestos en el equipo de desarrollo, de acuerdo con los roles que desempeñan en RUP.

| Puesto | Responsabilidad |
|---|---|
| **Jefe de Proyecto** | El jefe de proyecto asigna los recursos, gestiona las prioridades, coordina las iteraciones con los clientes y usuarios, y mantiene al equipo del proyecto enfocado en los objetivos. El jefe de proyecto también establece un conjunto de prácticas que aseguran la integridad y calidad de los artefactos del proyecto. Gestión de riesgos, planificación y control del proyecto. |
| **Arquitecto de Software** | Se encargará de supervisar el establecimiento de la Arquitectura del Sistema, es decir, definir la vista arquitectónica, los estilos arquitectónicos, el patrón de arquitectura y la arquitectura tecnológica a utilizar. Establecer la conectividad entre las diferentes sucursales de la empresa. |
| **Ingeniero de Software** | Gestión de requisitos, gestión de configuración y cambios, elaboración del modelo de datos, preparación de las pruebas funcionales, elaboración de la documentación. Elaborar modelos de implementación y despliegue. Captura, especificación y validación de requisitos, interactuando con el cliente y los usuarios mediante entrevistas. Elaboración del Modelo de Análisis y Diseño. Colaboración en la elaboración de las pruebas funcionales y el modelo de datos. Encargado además de la puesta en producción de la empresa. |
| **Programador** | Construcción de prototipos. Colaboración en la elaboración de las pruebas funcionales, modelo de datos y en las validaciones con el usuario. |
| **Téster** | Se encargará de la realización de las pruebas funcionales, de conectividad y rendimiento del sistema. |

#### *4.2.1.4 Gestión del proceso*.

##### *4.2.1.4.1 Estimaciones del proyecto.*

El presupuesto del proyecto y los recursos involucrados se adjuntan en un documento por separado.

*4.2.1.4.2 Plan del proyecto.*

En esta sección se presenta la organización del proyecto en fases e iteraciones y el calendario del mismo.

*4.2.1.4.2.1 Plan de fases.*

El desarrollo se llevará a cabo en base a fases con una o más iteraciones en cada una de ellas. La tabla 51 muestra la distribución de tiempos y el número de iteraciones de cada fase.

| Fase | Nro. Iteraciones | Duración |
|---|---|---|
| Fase de Inicio | 1 | 3 semanas |
| Fase de Elaboración | 2 | 2 semanas |
| Fase de Construcción | 3 | 7 semanas |
| Fase de Transición | 2 | 2 semanas |

***Tabla 51: Plan de Fases del Proyecto (*)***

Los hitos que marcan el final de cada fase se describen en la tabla 52.

| Descripción | Hito |
|---|---|
| Fase de Inicio | En esta fase se desarrollarán los requisitos del producto desde la perspectiva del usuario, los cuales serán establecidos en el artefacto Visión. Los principales casos de uso serán identificados y se hará un refinamiento al Plan de Desarrollo de Software. La aceptación del cliente/usuario del artefacto Visión y el Plan de Desarrollo de Software marcan el final de esta fase. |
| Fase de Elaboración | En esta fase se analizan los requisitos y se desarrolla un prototipo de arquitectura (incluyendo las partes más relevantes y/o críticas del sistema). Al final de esta fase, todos los casos de uso correspondientes a requisitos que serán implementados en la |

| | |
|---|---|
| | primera release de la fase de Construcción deben ser analizados y diseñados (en el Modelo de Análisis/Diseño). La revisión y aceptación del prototipo de la arquitectura del sistema marca el final de esta fase. |
| Fase de Construcción | Durante la fase de construcción se terminan de analizar y diseñar todos los casos de uso, refinando el Modelo de Análisis/Diseño. El producto se construye en base a dos iteraciones, cada una produciendo un release a la cual se le aplican las pruebas y se valida con el cliente/usuario. Se comienza la elaboración del material de apoyo al usuario. El hito que marca el fin de esta fase es la versión de la release, con la capacidad operacional parcial del producto que se haya considerado como crítica, lista para ser entregada a los usuarios para la prueba beta. |
| Fase de Transición | En esta fase prepararán dos release para distribución, asegurando una implantación y cambio previo del sistema de manera adecuada, incluyendo el entrenamiento de los usuarios. El hito que marca el fin de esta fase incluye, la entrega de toda la documentación del proyecto con los manuales de instalación y todo el material de apoyo al usuario, la finalización del entrenamiento de los usuarios y el empaquetado del producto. |

***Tabla 52: Hitos de las fases del Proyecto (*)***

*4.2.1.4.2.2 Calendario del proyecto.*

A continuación se muestra un calendario de las principales tareas del proyecto. El proceso iterativo e incremental de RUP se caracteriza por la realización en paralelo de todas las disciplinas de desarrollo a lo largo del proyecto, con lo cual la mayoría de los artefactos son generados tempranamente en el proyecto pero van desarrollándose en mayor o menor grado de acuerdo a la fase e iteración del proyecto.

La figura 29 ilustra este enfoque, en ella lo ensombrecido marca el énfasis de cada disciplina (workflow) en un momento determinado del desarrollo.

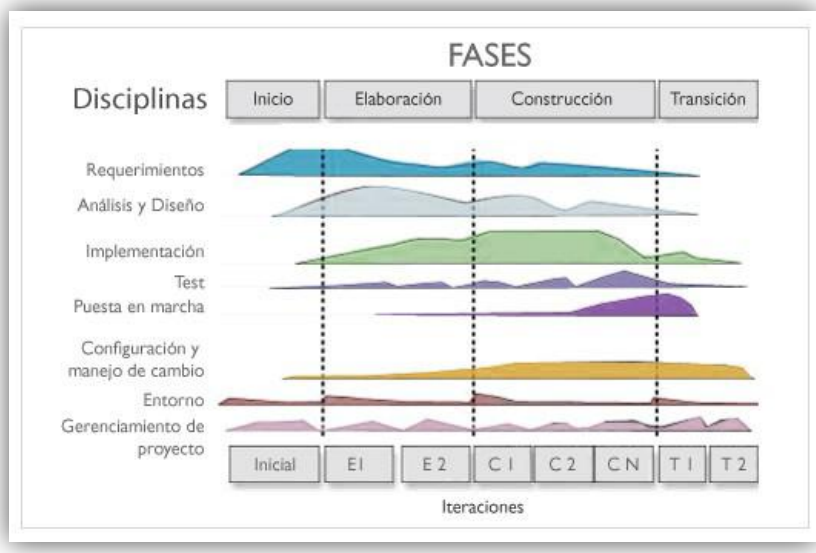


***Figura 29: Iteraciones del Proyecto***

Para este proyecto se ha establecido el siguiente calendario. La fecha de aprobación indica cuándo el artefacto en cuestión ha alcanzado un estado suficiente para ser sometido a revisión y aprobación, pero eso no quita la posibilidad de su posterior refinamiento y cambios.

| **Disciplinas/Artefactos generados o modificados durante la Fase de Inicio** | **Comienzo** | **Aprobación** |
|---|---|---|
| **Modelado del Negocio** | | |
| Modelo de Casos de Uso del Negocio y Modelado de Objetos del Negocio | Semana 04/06/2012 – 08/06/2012 | Semana 18/06/2012 – 22/06/2012 |
| **Requisitos** | | |
| Glosario | Semana 04/06/2012 – 08/06/2012 | Semana 18/06/2012 – 22/06/2012 |
| Visión | Semana 11/06/2012 – 15/06/2012 | Semana 18/06/2012 – 22/06/2012 |

| | | |
|---|---|---|
| Modelo de Casos de Uso | Semana 18/06/2012 – 22/06/2012 | Siguiente fase |
| Especificación de Casos de Uso | Semana 18/06/2012 – 22/06/2012 | Siguiente fase |
| Especificaciones Adicionales | Semana 18/06/2012 – 22/06/2012 | Siguiente fase |
| **Análisis/Diseño** | | |
| Modelado de Análisis/Diseño | Semana 11/06/2012 – 15/06/2012 | Siguiente fase |
| Modelo de Datos | Semana 11/06/2012 – 15/06/2012 | Siguiente fase |
| **Implementación** | | |
| Prototipos de Interfaces de Usuario | Semana 18/06/2012 – 22/06/2012 | Siguiente fase |
| Modelo de Implementación | Semana 18/06/2012 – 22/06/2012 | Siguiente fase |
| **Pruebas** | | |
| Casos de Prueba Funcionales | Semana 18/06/2012– 22/06/2012 | Siguiente fase |
| **Despliegue** | | |
| Modelo de Despliegue | Semana 18/06/2012– 22/06/2012 | Siguiente fase |
| Gestión de Cambios y Configuración | Durante todo el Proyecto | Durante todo el Proyecto |
| **Gestión del Proyecto** | | |
| Plan de Desarrollo de Software en su versión 1.0 y planes de las Iteraciones | Semana 04/06/2012– 08/06/2012 | Semana 18/06/2012 – 22/06/2012 |
| Ambiente | Durante todo el Proyecto | Durante todo el Proyecto |

***Tabla 53: Calendario del Proyecto: Fase de Inicio (*)***

| Disciplinas/Artefactos generados o modificados durante la Fase de Elaboración | Comienzo | Aprobación |
|---|---|---|
| **Modelado del Negocio** | | |
| Modelo de Casos de Uso del Negocio y Modelado de Objetos del Negocio | Semana 04/06/2012 – 08/06/2012 | Aprobado |
| **Requisitos** | | |
| Glosario | Semana 04/06/2012 – 08/06/2012 | Aprobado |
| Visión | Semana 11/06/2012 – 15/06/2012 | Aprobado |
| Modelo de Casos de Uso | Semana 18/06/2012 – 22/06/2012 | Semana 05/06/2012 – 09/06/2012 |
| Especificación de Casos de Uso | Semana 18/06/2012 – 22/06/2012 | Semana 05/06/2012 – 09/06/2012 |
| Especificaciones Adicionales | Semana 18/06/2012 – 22/06/2012 | Semana 05/06/2012 – 09/06/2012 |
| **Análisis/Diseño** | | |
| Modelado de Análisis/Diseño | Semana 11/06/2012 – 15/06/2012 | Revisar en cada Iteración |
| Modelo de Datos | Semana 11/06/2012 – 15/06/2012 | Revisar en cada Iteración |
| **Implementación** | | |
| Prototipos de Interfaces de Usuario | Semana 18/06/2012 – 22/06/2012 | Revisar en cada Iteración |
| Modelo de Implementación | Semana 18/06/2012 – 22/06/2012 | Revisar en cada Iteración |
| **Pruebas** | | |
| Casos de Prueba Funcionales | Semana 18/06/2012 – 22/06/2012 | Revisar en cada Iteración |
| **Despliegue** | | |
| Modelo de Despliegue | Semana 18/06/2012 – 22/06/2012 | Revisar en cada Iteración |
| Gestión de Cambios y Configuración | Durante todo el Proyecto | Durante todo el Proyecto |
| **Gestión del Proyecto** | | |
| Plan de Desarrollo de Software en su versión 1.0 y planes de las Iteraciones | Semana 04/06/2012 – 08/06/2012 | Revisar en cada Iteración |
| Ambiente | Durante todo el Proyecto | Durante todo el Proyecto |

***Tabla 54: Calendario del Proyecto: Fase de Elaboración (*)***

| Disciplinas/Artefactos generados o modificados durante la Fase de Construcción (Iteración 1) | Comienzo | Aprobación |
|---|---|---|
| **Casos de Uso negociados para la Primera Release** | | |
| Gestión de Clientes | 09/07/2012 | 06/08/2012 |
| Buscar Cliente | 09/07/2012 | 06/08/2012 |
| Gestión de Habitaciones | 16/07/2012 | 06/08/2012 |
| Buscar Habitación | 16/07/2012 | 06/08/2012 |
| Manejo de Usuarios | 23/07/2012 | 06/08/2012 |
| Buscar Usuario | 23/07/2012 | 06/08/2012 |
| Manejo de Roles | 30/07/2012 | 06/08/2012 |
| Buscar Rol | 30/07/2012 | 06/08/2012 |

***Tabla 55: Calendario del Proyecto: Fase de Construcción (Iteración 1) ****

| Disciplinas/Artefactos generados o modificados durante la Fase de Construcción (Iteración 2) | Comienzo | Aprobación |
|---|---|---|
| **Casos de Uso negociados para la Primera Release** | | |
| Gestión de Clientes | 09/07/2012 | Aprobado |
| Buscar Cliente | 09/07/2012 | Aprobado |
| Gestión de Habitaciones | 16/07/2012 | Aprobado |
| Buscar Habitación | 16/07/2012 | Aprobado |
| Manejo de Usuarios | 23/07/2012 | Aprobado |
| Buscar Usuario | 23/07/2012 | Aprobado |
| Manejo de Roles | 30/07/2012 | Aprobado |
| Buscar Rol | 30/07/2012 | Aprobado |
| **Casos de Uso negociados para la Segunda Release** | | |
| Gestión de Reservaciones | 13/08/2012 | 17/09/2012 |
| Buscar Reserva | 13/08/2012 | 17/09/2012 |
| Facturación | 20/08/2012 | 17/09/2012 |
| Login – Logout | 03/09/2012 | 17/09/2012 |
| Gestionar Reportes | 10/09/2012 | 17/09/2012 |

***Tabla 56: Calendario del Proyecto: Fase de Construcción (Iteración 2) ****

*4.2.1.4.2.3 Seguimiento y control del proyecto.*

**Gestión de Requisitos**

Los requisitos del sistema son especificados en el artefacto Visión. Cada requisito tiene una serie de atributos como importancia, estado, iteración, dónde se implementa, etc. Estos atributos permitirán realizar un seguimiento garantizado de cada requisito. Los cambios en los requisitos serán administrados mediante una Solicitud de Cambio, las cuales serán evaluadas y distribuidas para asegurar la integridad del sistema y el correcto proceso de gestión y configuración de cambios.

**Control de Plazos**

El calendario del proyecto tendrá un seguimiento y evaluación semanal por el jefe de proyecto.

**Gestión de Riesgos**

A partir de la fase de Inicio se mantendrá una lista de riesgos asociados al proyecto y a las acciones establecidas como estrategia para mitigarlos o acciones de contingencia. Esta lista será evaluada al menos una vez cada iteración.

**Gestión de Configuración**

Se realizará una gestión de configuración para llevar un registro de los artefactos generados y sus versiones. También se incluirá la administración de las Solicitudes de Cambio y de las modificaciones que éstas produzcan, informando y publicando dichos cambios para que sean accesibles a todos los participantes en el proyecto.

Al final de cada iteración se establecerá un baseline (un registro del estado de cada artefacto, estableciendo una versión), la cual podrá ser modificada solo por una Solicitud de Cambio aprobada.

## 4.3 Modelado del Negocio

A continuación se presentan los modelos definidos en RUP como modelo del negocio, modelo de datos y modelo de análisis y diseño.

**RESERVACIÓN:**

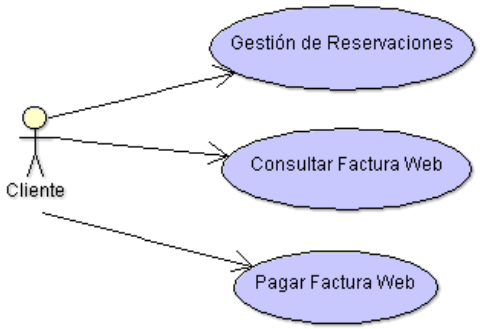


***Figura 30: Diagrama de Caso de Uso del Módulo de Reservación (*)***

**RECEPCIÓN:**

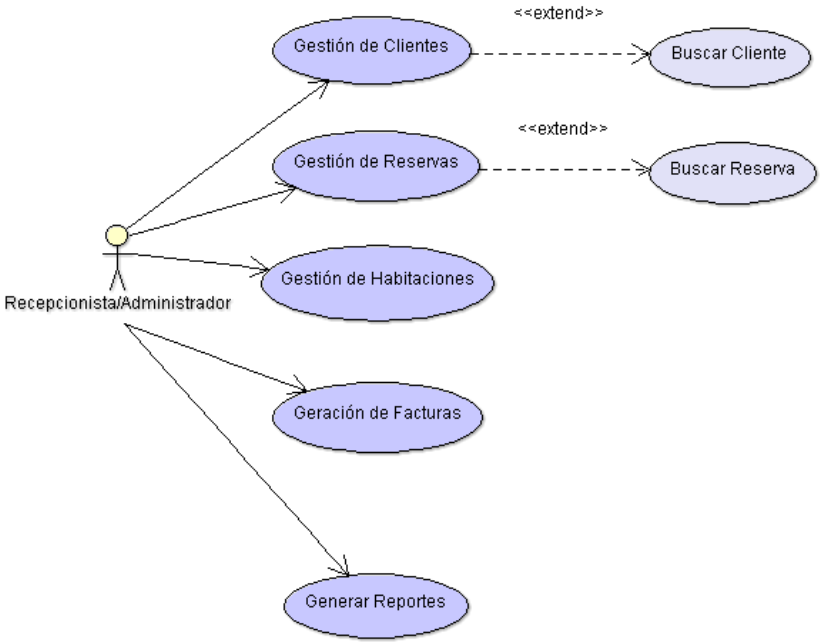


***Figura 31: Diagrama de Caso de Uso del Módulo de Recepción (*)***

**SEGURIDAD:**

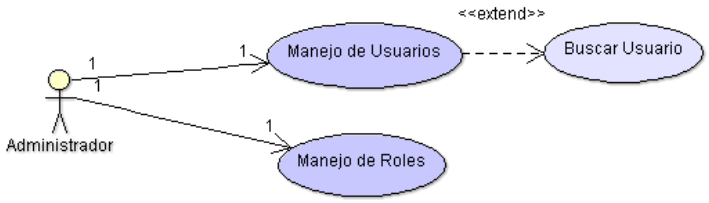


***Figura 32: Diagrama de Caso de Uso del Módulo de Seguridad (*)***

**GENERAL:**

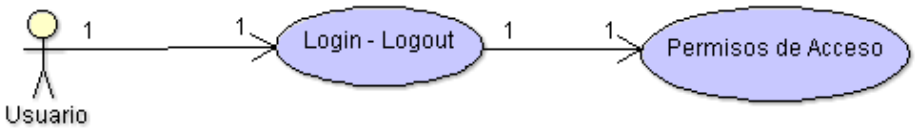


***Figura 33: Diagrama de Caso de Uso General (*)***

## 4.4 Requisitos

A continuación se muestran los artefactos utilizados para declarar los requisitos del software, es decir, el documento visión, el documento glosario y las especificaciones de los casos de uso.

### 4.4.1 Visión.

#### *4.4.1.1 Introducción.*

El presente documento es uno de los artefactos que se encuentran dentro de la metodología RUP, se centra en la funcionalidad requerida por los participantes del proyecto y los usuarios finales.

*4.4.1.1.1 Propósito.*

El propósito de este documento es recoger, analizar y definir las necesidades de alto nivel y las características del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro.

Los detalles de cómo el sistema cubre los requerimientos se pueden observar en la especificación de los casos de uso y otros documentos adicionales.

*4.4.1.1.2 Alcance.*

El documento Visión se ocupa, como ya se ha descrito, del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro. Dicho sistema será desarrollado por Maricruz Acosta Yerovi.

El sistema permitirá a los encargados de la empresa, controlar las actividades correspondientes a la reservación de habitaciones, generación de facturas y pago on-line a través del uso de PayPal.

***4.4.1.2 Posicionamiento.***

*4.4.1.2.1 Oportunidad de negocio.*

El sistema planteado permitirá a la empresa el control de las actividades relacionadas con las reservaciones efectuadas no solo en el complejo, sino que también pueden ser realizadas vía Web, lo cual supondrá un acceso rápido y sencillo a los datos, gracias a la implementación de interfaces gráficas sencillas y amigables. Además, los datos accedidos estarán siempre actualizados, lo cual constituye un factor importante para lograr un control centralizado de la información.

#### *4.4.1.2.2 Sentencia que define el problema.*

| | |
|---|---|
| **El problema de** | La Empresa Pública Santa Agua de Chachimbiro no cuenta con un sistema que permita realizar reservaciones y emita la factura correspondiente. |
| **Afecta a** | Los clientes y personal de la Empresa involucrados en los diferentes procesos relacionados con la gestión de reservaciones y cobros. |
| **El impacto asociado es** | Efectuar reservaciones físicas y a través del Internet con la emisión de la respectiva factura, contando así con información actualizada que permita la generación oportuna de reportes. |
| **Una solución adecuada sería** | Crear una aplicación Web vinculada al Sitio Web de la Empresa, con el uso de herramientas libres y una base de datos accesible; permitiendo gestionar los procesos de reserva y cobro, y, la generación de interfaces amigables y sencillas que accedan a dicha base de datos. |
| **Para** | Stakeholder de la Empresa,<br>Personal encargado de reservaciones y cobros,<br>Usuarios externos o clientes. |
| **Quienes** | Intervienen en los procesos de la Empresa. |
| **El nombre del producto** | Sistema hotelero. |
| **Que** | Almacena la información necesaria para gestionar reservaciones físicas y vía Web. |
| **No como** | El sistema actual. |
| **Nuestro producto** | Permite la administración de las actividades correspondientes a la reservación de habitaciones (cabañas), generación de facturas y pago on-line a través del uso de PayPal, mediante una una interfaz gráfica, sencilla y amigable que proporciona un acceso rápido y actualizado de la información desde cualquier lugar que tenga acceso a Internet. |

***Tabla 57: Sentencia que define el problema del Proyecto (*)***

#### *4.4.1.3 Descripción de Stakeholders (participantes en el proyecto) y usuarios.*

Para proveer de forma efectiva productos y servicios que se ajusten a las necesidades de los usuarios, es necesario identificar e involucrar a todos los participantes en el proyecto como parte del proceso de modelado de requerimientos.

También es necesario identificar los usuarios del sistema y asegurarse de que el conjunto de participantes en el proyecto, los representa adecuadamente.

Esta sección muestra un perfil de los participantes y de los usuarios involucrados en el proyecto, así como los problemas más importantes que estos perciben para enfocar la solución propuesta hacia ellos. No describe sus requisitos específicos ya que éstos se capturan con otro artefacto. En lugar de esto proporciona la justificación del por qué estos requisitos son necesarios.

*4.4.1.3.1 Resumen de los Stakeholders.*

| Nombre | Descripción | Responsabilidades |
|---|---|---|
| Gerente de la empresa | Representante global de la empresa. | El Stakeholder:<br>• Representa a todos los usuarios posibles del sistema.<br>• Realiza el seguimiento del desarrollo del proyecto.<br>• Aprueba requisitos y funcionalidades. |

***Tabla 58: Resumen de los Stakeholders del Proyecto (*)***

*4.4.1.3.2 Resumen de los usuarios.*

| Nombre | Descripción | Stakeholder |
|---|---|---|
| ACT1 Administrador | Responsable de la gestión de usuarios, roles y la administración del sistema. | STK1 Seguridad |
| ACT2 Recepcionista | Encargado de la facturación y gestión de reservaciones y clientes. | STK2 Recepción, STK3 Facturación. |
| ACT3 Usuario externo, Cliente | Realiza reservaciones a través de Internet. | STK4 Reservación |

***Tabla 59: Resumen de los Usuarios del Proyecto (*)***

*4.4.1.3.3 Entorno de usuario.*

Los usuarios ingresarán al sistema identificándose y tras este paso, accederán a la parte de la aplicación diseñada para cada uno según su papel. Necesitan disponer únicamente de un navegador web y acceso a internet. Los reportes pueden ser generados en pdf y/o Microsoft Excel.

*4.4.1.3.4 Perfil de los stakeholders.*

**REPRESENTANTE DEL ÁREA TÉCNICA Y SISTEMAS DE INFORMACIÓN**

| | |
|---|---|
| **Representante** | Gerente de la empresa. |
| **Descripción** | Representante global de la empresa. |
| **Tipo** | Experto en negocios. |
| **Responsabilidades** | Responsable de indicar las necesidades de los usuarios del sistema. También realiza el seguimiento al desarrollo del proyecto y la aprobación de los requerimientos y funcionalidades del sistema. |
| **Criterio de éxito** | A definir por el cliente. |
| **Grado de participación** | Revisión de requerimientos y estructura del sistema. |

***Tabla 60: Representante del Área Técnica del Proyecto (*)***

#### *4.4.1.3.5 Perfiles de usuario.*

### ADMINISTRADOR

| | |
|---|---|
| **Representante** | STK1 SEGURIDAD |
| **Descripción** | Administrador |
| **Tipo** | Experto en sistemas. |
| **Responsabilidades** | Responsable de la gestión de usuarios, roles y la administración del sistema. |
| **Criterio de éxito** | A definir por el cliente. |
| **Grado de participación** | A definir por el cliente. |

***Tabla 61: Usuario Administrador del Proyecto (*)***

### RECEPCIONISTA

| | |
|---|---|
| **Representante** | STK2 RECEPCIÓN y STK3 FACTURACIÓN |
| **Descripción** | Recepcionista |
| **Tipo** | Usuario experto |
| **Responsabilidades** | Encargado de la facturación y de la administración de reservaciones y clientes. |
| **Criterio de éxito** | A definir por el cliente. |
| **Grado de participación** | A definir por el cliente. |

***Tabla 62: Usuario Recepcionista del Proyecto (*)***

### CLIENTE

| | |
|---|---|
| **Representante** | STK4 RESERVACIÓN |
| **Descripción** | Cliente de la Empresa |
| **Tipo** | Usuario externo |
| **Responsabilidades** | Realiza el proceso de reservaciones a través de Internet. |
| **Criterio de éxito** | A definir por el cliente. |
| **Grado de participación** | A definir por el cliente. |

***Tabla 63: Usuario Cliente del Proyecto (*)***

#### *4.4.1.4 Descripción global del producto.*

##### *4.4.1.4.1 Perspectiva del producto.*

El producto a desarrollar es un sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro, con el fin de mejorar el proceso de reservas actual. Las áreas a tratar por el sistema son: reservación on-line y física, facturación, reportes y seguridad.

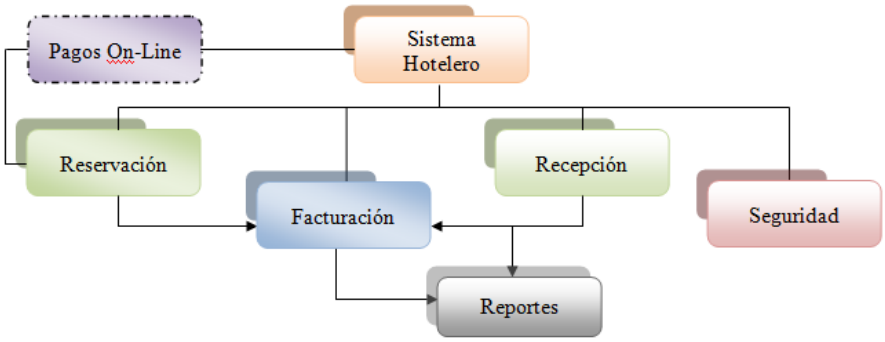


***Figura 34: Módulos del Sistema hotelero (*)***

##### *4.4.1.4.2 Resumen de las características.*

A continuación se señala un listado de los beneficios que obtendrá el cliente con la utilización del producto.

| Beneficio del cliente | Características que lo apoyan |
|---|---|
| Gestión y agilidad en el proceso de reservaciones, que pueden ser realizadas también vía Web. | Aplicación Web desde la cual se pueden realizar reservaciones. |
| Mayor facilidad para la administración de clientes y personal. | Base de datos centralizada que maneja información de los usuarios. |
| Mayor control de las reservaciones. | El sistema gestionará las reservaciones realizadas, generará reportes. |
| Seguridad. | El ingreso al sistema es controlado mediante un usuario y contraseña y el acceso a las opciones por medio de roles y permisos. |

***Tabla 64: Resumen de las características del Proyecto (*)***

*4.4.1.4.3 Suposiciones y dependencias.*

El sistema hotelero de Santa Agua de Chachimbiro es una aplicación Web que va a ser integrada al sitio web de la Empresa, la cual está alojada en el servidor del Gobierno Provincial de Imbabura.

Con el propósito de garantizar el término de una transacción de reserva, se pretende minimizar el uso de gráficos que impidan el uso ágil de la página. Todo término de transacción satisfactorio o insatisfactorio será notificado de forma inmediata al usuario.

### *4.4.1.5 Descripción global del producto.*

**CSW1: RESERVACIÓN**

Los clientes registrados obtendrán acceso al módulo de *Reservación*, encargado del proceso de reservas vía web, controla la disponibilidad de las cabañas, permite el registro del cliente y ofrece el servicio de pagos on-line a través de PayPal.

**CSW2: RECEPCIÓN**

La persona encargada de receptar las reservaciones, podrá acceder al módulo de *Recepción* que gestiona el proceso de reservaciones en las instalaciones de la Empresa, realiza el registro de huéspedes con o sin reservación, modificación de los datos de reservaciones y consultas.

**CSW3: FACTURACIÓN**

El recepcionista hará uso del módulo de *Facturación* que emite la factura por concepto de pago de hospedaje, una vez que se haya confirmado la recepción. El cliente que utiliza el módulo de Reservación podrá visualizar la factura para la cancelación del valor correspondiente.

**CSW4: SEGURIDAD**

El administrador del sistema será el encargado del manejo el módulo de *Seguridad* que administra el acceso al sistema y gestiona usuarios.

#### *4.4.1.6 Restricciones.*

Las restricciones que el sistema presenta y advierte a sus usuarios es la necesidad de contar con acceso a Internet y un navegador web (firefox, opera, safari, internet explorer, etc).

El hardware requerido es el adecuado, se cuenta con un servidor para el alojamiento del servidor de aplicaciones y del sistema de gestión de base de datos relacional.

#### *4.4.1.7 Otros requisitos del proyecto.*

##### *4.4.1.7.1 Estándares aplicables.*

Lenguaje para el diseño de páginas WEB: HTML avanzado (compatible desde cualquier navegador web).
Sistema de Gestión de Base de Datos Relacional: PostgreSQL.
Protocolo de Comunicación: TCP/IP versión 4.

##### *4.4.1.7.2 Requisitos de sistema.*

Como ya se menciono con anterioridad, el requerimiento esencial para los usuarios es contar con una conexión adecuada a internet, y un navegador web (firefox, opera, safari, internet explorer, etc).

##### *4.4.1.7.3 Requisitos de desempeño.*

El mayor requerimiento de rendimiento estará determinado por la facilidad de acceso al sistema de la empresa. Se ha considerado un diseño vistoso pero ligero de peso.

##### *4.4.1.7.4 Requisitos de entorno.*

Los dispositivos de red y servidores, tendrán que ser fijados en un rack para cumplir con algunos estándares de cableado estructurado. Se necesita un ambiente con temperaturas: *-5<15<+5 °C*.

#### *4.4.1.8 Requisitos de documentación.*

##### *4.4.1.8.1 Manual de usuario.*

Los manuales de usuario podrán ser descargados directamente desde el sistema. El manual de usuario contendrá la documentación de la instalación, uso y mantenimiento del sistema. Estará enfocado en cada rol de usuario, ya que cada uno maneja diferentes tipos de información.

##### *4.4.1.8.2 Ayuda en línea.*

Se podrá acceder a la ayuda en línea por medio de la utilización de los diferentes hipervínculos situados cerca de las opciones relevantes del sistema.

##### *4.4.1.8.3 Guías de instalación y configuración.*

Esta información se encuentra en los anexos digitales del proyecto.

### 4.4.2 Glosario.

#### *4.4.2.1 Introducción.*

Este documento recoge cada uno de los términos manejados a lo largo del proyecto de desarrollo del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro. Se trata de un diccionario informal de datos y definiciones de la nomenclatura que es manejada, de tal modo que se crea un estándar para todo el proyecto.

*4.4.2.1.1 Propósito.*

El propósito de este glosario es definir con exactitud y sin ambigüedad la terminología manejada en el proyecto de desarrollo del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro. También sirve como guía de consulta para la clarificación de los puntos conflictivos o poco esclarecedores del proyecto.

*4.4.2.1.2 Alcance.*

El alcance del presente glosario se extiende a todos los módulos definidos en el sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro. De manera que la terminología empleada por las diferentes divisiones de la empresa; se refleja con claridad aquí.

***4.4.2.2 Organización del glosario.***

El presente glosario está organizado por definiciones de términos ordenados de forma ascendente según la ordenación alfabética tradicional.

***4.4.2.3 Definiciones***.

A continuación se indican los términos manejados a lo largo de todo el proyecto de desarrollo del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro.

**ARTEFACTO:**

Los artefactos son los productos tangibles del proceso por ejemplo, modelo de casos de uso y código fuente.

**CLIENTE EXTERNO:**

El cliente externo es el cliente propiamente dicho, es decir, la visión que ofrece RUP del modelo de caso de uso del negocio, el cliente externo representa uno de los tantos agentes externos con los que interactúa la empresa. Por lo tanto, el cliente externo es cualquier persona que hará uso de las instalaciones del complejo.

**IDE (INTEGRATE DEVELOPMENT ENVIROMENT):**

Entorno de desarrollo, diseño integrado o entorno de depuración integrada; proporciona las instalaciones para el desarrollo del software: Editor de código fuente, compilador y herramientas de automatización.

**LÓGICA DEL NEGOCIO:**

Gran cantidad de información requiere ser tratada por una lógica específica para poder convertirse en información útil para el usuario.

**POJO (PLAIN OLD JAVA OBJECT):**

Enfatiza el uso de clases simples en el marco de una revalorización de la programación.

**ROL:**

Papel que desempeña una persona en un determinado momento, una persona puede desempeñar distintos roles a lo largo del proceso.

**TRANSFORMACIÓN DE DATOS:**

Permite operaciones sofisticadas en agrupación de cantidades, porcentajes de los totales generales y más. Datos ordenados, filtrados, según la necesidad del usuario.

### 4.4.3 Especificación del caso de uso: "Gestionar Reservaciones".

Este caso de uso permite que el cliente realice la administración de la reservación efectuada por él; el usuario podrá crear, modificar o eliminar la reservación según sea el criterio o necesidad a satisfacer.

#### *4.4.3.1 Flujo de eventos.*

*4.4.3.1.1 Flujos básicos.*

El sistema muestra la interfaz *"Gestión de Reservaciones"* con los criterios de búsqueda que es el campo código reserva; además de las opciones buscar, editar, nuevo, eliminar.

**CREAR RESERVACIÓN**

1. El caso de uso comienza cuando el cliente solicita *"Crear reservación"* en la interfaz *"Gestión de Reservaciones".*
2. El sistema muestra la interfaz *"Crear reservación".*
3. El cliente ingresa los datos para realizar la reservación.
4. El cliente elige la opción *"Realizar reserva"* para guardar la reservación.
5. El sistema guarda registro nuevo de la reservación realizada.
6. El sistema genera mensaje de confirmación de creación de reserva.
7. El sistema regresa a la interfaz *"Gestión de Reservaciones".*

**MODIFICAR RESERVACIÓN**

*1.* El cliente ingresa el criterio de búsqueda de la reservación en la interfaz *"Gestión de Reservaciones".*

2. El cliente selecciona el botón *"Buscar".*
3. Se ejecuta el caso de uso extendido *"Buscar Reservación"*.
4. El sistema muestra las reservaciones realizadas y sus datos en una tabla en la interfaz *"Gestión de Reservaciones".*
5. El cliente presiona *"Editar reservación"* en la reservación que desea modificar.
6. El cliente modifica los datos de la reservación según su criterio.
7. El cliente elige la opción *"Guardar cambios"*.
8. El sistema guarda los datos modificados y genera el mensaje *"Modificación realizada".*
9. El sistema regresa a la interfaz *"Gestión de Reservaciones".*

**ELIMINAR RESERVACIÓN**

1. El cliente ingresa el criterio de búsqueda de la reservación en la interfaz *"Gestión de Reservaciones".*
2. El cliente selecciona el botón *"Buscar".*
3. Se ejecuta el caso de uso extendido *"Buscar Reservación"*.
4. El sistema muestra las reservaciones realizadas y sus datos en una tabla en la interfaz *"Gestión de Reservaciones".*
5. El cliente selecciona la reservación que desea eliminar.
6. El cliente presiona el botón *"Eliminar reserva"*.
7. El sistema genera el mensaje *"Reservación eliminada".*
8. El sistema elimina la reservación seleccionada de la base de datos.

*4.4.3.1.2 Flujos alternativos.*

**ERROR FALTA DE INGRESAR DATOS OBLIGATORIOS**

En el punto cuatro de crear reservación, cuando el usuario selecciona *"Realizar reserva"* sin haber llenado todos los campos requeridos, el sistema muestra un mensaje de error "Campo requerido" en cada uno de los campos que son necesarios.

**NO COLOCA CRITERIO DE BÚSQUEDA**

Si en modificar reservación y eliminar reservación el cliente presiona buscar sin ingresar un criterio de búsqueda, el sistema mostrará un mensaje "Campo requerido".

#### *4.4.3.2 Precondiciones.*

El cliente debe estar registrado en el sistema.

Deben existir reservaciones en la base de datos.

#### *4.4.3.3 Postcondiciones.*

El sistema ha actualizado la lista de reservaciones.

#### *4.4.3.4 Puntos de extensión.*

El sistema llama al caso de uso *"Buscar Reservación"*.

### 4.4.4 Especificación del caso de uso: "Login – Logout".

Este caso de uso permite al usuario ingresar al sistema para poder obtener los privilegios necesarios para manejar el sistema y cerrar su sesión, una vez concluida la realización de sus operaciones.

#### *4.4.4.1 Flujo de eventos*.

##### *4.4.4.1.1 Flujo básicos.*

**LOGIN**

1. El sistema solicita la cuenta de usuario.
2. El usuario ingresa su cuenta de usuario en la interfaz *"Autentificación de Usuarios"*.
3. EL sistema solicita contraseña de usuario.
4. El usuario ingresa su contraseña de usuario en la interfaz *"Autentificación de Usuarios"*.
5. El usuario selecciona la opción *"Aceptar"* y el caso de uso finaliza.

**LOGOUT**

1. El usuario selecciona la opción *"Cerrar Sesión"*.
2. El sistema solicita la confirmación del cierre de sesión.
3. Se cierra la sesión y se muestra la interfaz *"Autentificación de Usuarios"*.

##### *4.4.4.1.2 Flujos alternativos.*

**ERROR DE CUENTA DE USUARIO**

En el punto dos de login, si el usuario ingresa de manera incorrecta su cuenta el sistema muestra un mensaje de error *"Usuario o contraseña incorrectos".*

**ERROR DE CONTRASEÑA**

En el punto cuatro de login, si el usuario ingresa de manera incorrecta su contraseña el sistema muestra un mensaje de error *"Usuario o contraseña incorrectos".*

#### *4.4.4.2 Precondiciones.*

El usuario debe tener una cuenta de usuario.

#### *4.4.4.3 Postcondiciones.*

El sistema queda conectando mediante una sesión y se genera el acceso a las opciones que el usuario tiene con sus privilegios.

### 4.4.5 Requerimientos.

#### *4.4.5.1 Stakeholders.*

Los representantes de los usuarios y portavoces de las necesidades de la empresa son los stakeholders. En este proyecto solamente se ha tratado con un stakeholder como representante de los usuarios y necesidades de la empresa, sin embargo se han dividido representativamente.

La matriz de atributos de los stakeholders es la siguiente:

| Requerimientos | Representante | Ubicación |
|---|---|---|
| STK1: Reservación | Cliente | Documento Visión |
| STK2: Recepción | Recepcionista/ Administrador | Documento Visión |
| STK3: Facturación | Recepcionista/ Administrador | Documento Visión |
| STK4: Seguridad | Administrador | Documento Visión |

***Tabla 65: Matriz de atributos de los Stackholders del Proyecto (*)***

### *4.4.5.2 Actores.*

Se define este requerimiento para listar los usuarios potenciales del sistema, en este proyecto se han definido los siguientes actores: *Cliente, Recepcionista y Administrador.*

| Requerimientos | Ubicación | Módulo |
|---|---|---|
| ACT1: Cliente | Documento Visión | Reservación |
| ACT2: Recepcionista | Documento Visión | Recepción y Facturación |
| ACT3: Administrador | Documento Visión | Seguridad |

***Tabla 66: Matriz de atributos de los Actores del Proyecto (*)***

### *4.4.5.3 Características de software.*

Las características de software son las necesidades de los usuarios propuestas por los stakeholders de la empresa, son los requisitos que debe cumplir el sistema para satisfacer las necesidades de los trabajadores y de la empresa.

Las características definidas son las que aparecen en la matriz de atributos, siendo indicadas como sub características derivadas según una clasificación jerárquica.

| Requerimientos | Asignado a |
|---|---|
| CSW1: Reservación<br>Reservación | |
| CSW1.1: Gestión de reservaciones<br>Gestión de reservaciones | Equipo completo de análisis, desarrollo e implementación |
| CSW2: Recepción<br>Recepción | |
| CSW2.1: Gestión de datos de clientes<br>Gestión de datos de clientes | Equipo completo de análisis, desarrollo e implementación |
| CSW2.2: Gestión de reservas<br>Gestión de reservas | Equipo completo de análisis, desarrollo e implementación |
| CSW2.3: Gestión de habitaciones<br>Gestión de habitaciones | Equipo completo de análisis, desarrollo e implementación |
| CSW3: Facturación<br>Facturación | |
| CSW3.1: Gestión de facturación<br>Gestión de facturación | Equipo completo de análisis, desarrollo e implementación |
| CSW4: Seguridad<br>Seguridad | |
| CSW4.1: Manejo de usuarios<br>Manejo de usuarios | Equipo completo de análisis, desarrollo e implementación |
| CSW4.2: Manejo de roles<br>Manejo de roles | Equipo completo de análisis, desarrollo e implementación |

***Tabla 67: Matriz de atributos de las características del Proyecto (*)***

### *4.4.5.4 Casos de uso.*

Derivados de las características de software, son el resultado del análisis de las necesidades de los usuarios. La matriz de atributos es la siguiente:

| Requerimientos | Asignado a |
|---|---|
| ECU1: Gestión de Reservaciones. | Equipo de desarrollo. |
| ECU2: Buscar Reservación. | Equipo de desarrollo. |
| ECU3: Consultar Factura. | Equipo de desarrollo. |
| ECU4: Pagar Factura Web. | Equipo de desarrollo. |
| ECU5: Gestión del Cliente. | Equipo de desarrollo. |

| | |
|---|---|
| ECU6: Buscar Cliente. | Equipo de desarrollo. |
| ECU7: Gestión de Reservas. | Equipo de desarrollo. |
| ECU8: Buscar Reserva. | Equipo de desarrollo. |
| ECU9: Gestión de Habitaciones. | Equipo de desarrollo. |
| ECU10: Gestión de Facturación. | Equipo de desarrollo. |
| ECU11: Manejo de Usuarios. | Equipo de desarrollo. |
| ECU12: Buscar Usuario. | Equipo de desarrollo. |
| ECU13: Manejo de Roles. | Equipo de desarrollo. |
| ECU14: Login - Logout. | Equipo de desarrollo. |
| ECU15: Generar Reportes. | Equipo de desarrollo. |

***Tabla 68: Matriz de atributos de los Casos de Uso del Proyecto (*)***

### *4.4.5.5 Clases.*

| **Requerimientos** | **Ubicación** |
|---|---|
| CLS1: Clientes | Base de Datos |
| CLS2: UsuarioEmpleado | Base de Datos |
| CLS3: Factura | Base de Datos |
| CLS4: Reserva | Base de Datos |
| CLS5: LineasReserva | Base de Datos |
| CLS6: Instalacion | Base de Datos |
| CLS7: Habitacion | Base de Datos |
| CLS8: TipoHabitacion | Base de Datos |
| CLS9: Planes | Base de Datos |
| CLS10: TipoPlan | Base de Datos |
| CLS11: Descuento | Base de Datos |
| CLS12: Pais | Base de Datos |

***Tabla 69: Matriz de atributos de las Clases del Proyecto (*)***

## 4.5 Análisis/Diseño

A continuación se presentan el modelo definido en RUP como modelo de análisis/diseño (Modelo Relacional).

### 4.5.1 Modelo Entidad-Relación.

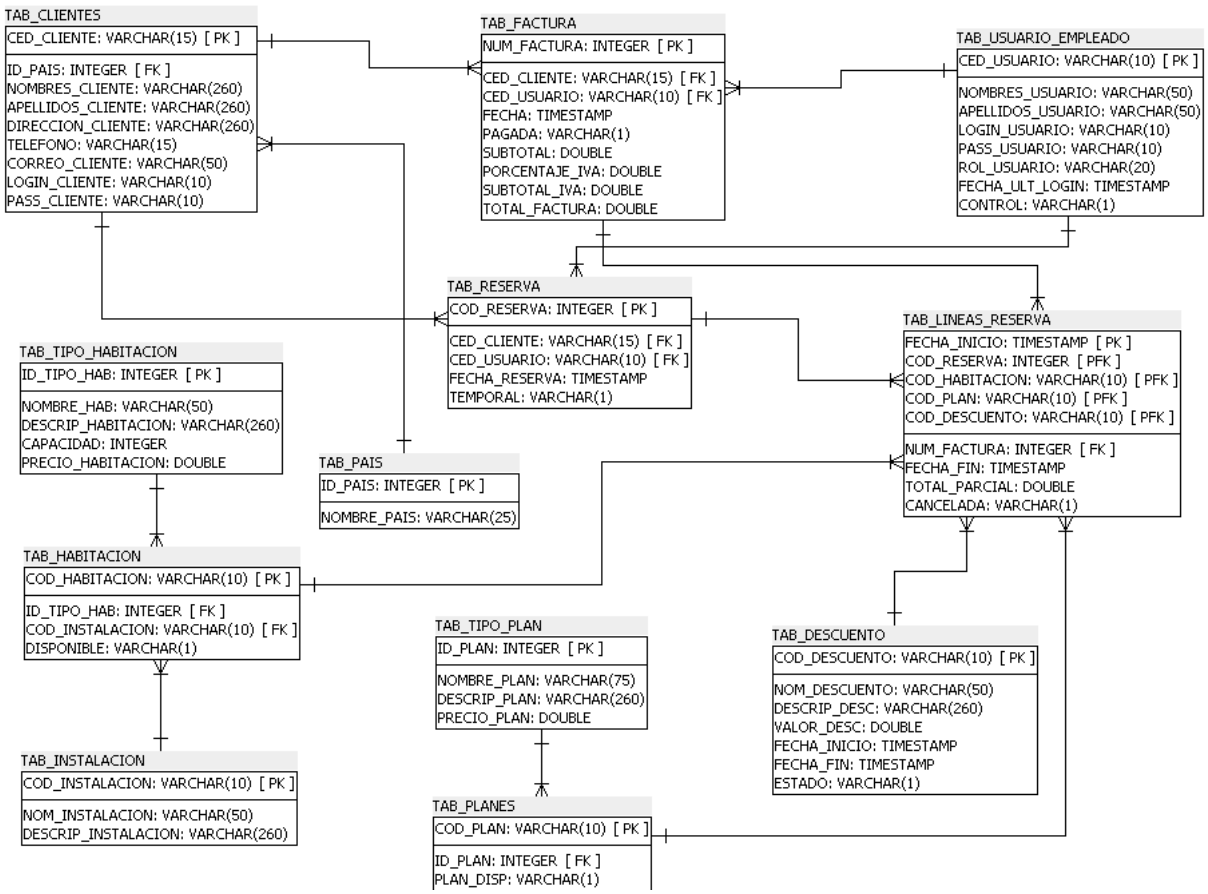


***Figura 35: Diagrama Relacional del sistema hotelero (*)***

## 4.6 Implementación

A continuación se presentan en los modelos definidos en RUP como prototipos de interfaces de usuario, diagrama de componentes y diagrama de despliegue del proyecto.

### 4.6.1 Prototipos de interfaces de usuario.

A continuación se presentan los prototipos de interfaces, gráficas de usuario diseñadas para la aplicación final.

**PRINCIPAL**


***Figura 36: Interfaz de Usuario: Login del Proyecto (*)***

**INGRESO DE RESERVA**


***Figura 37: Interfaz de Usuario: Ingreso de Reserva (*)***

### 4.6.2 Diagrama de componentes.

Se muestra la disposición de las partes integrantes de la aplicación y las dependencias entre los distintos módulos de la aplicación.

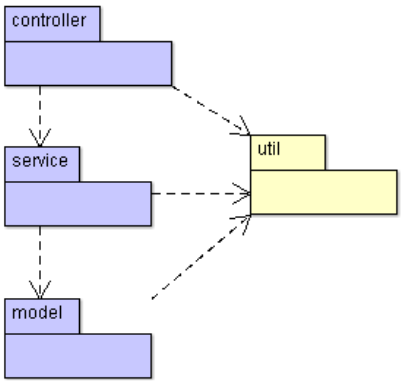


***Figura 38: Diagrama de Componentes del Proyecto (*)***

### 4.6.3 Diagrama de despliegue.

Se muestra la disposición de la arquitectura de la aplicación.

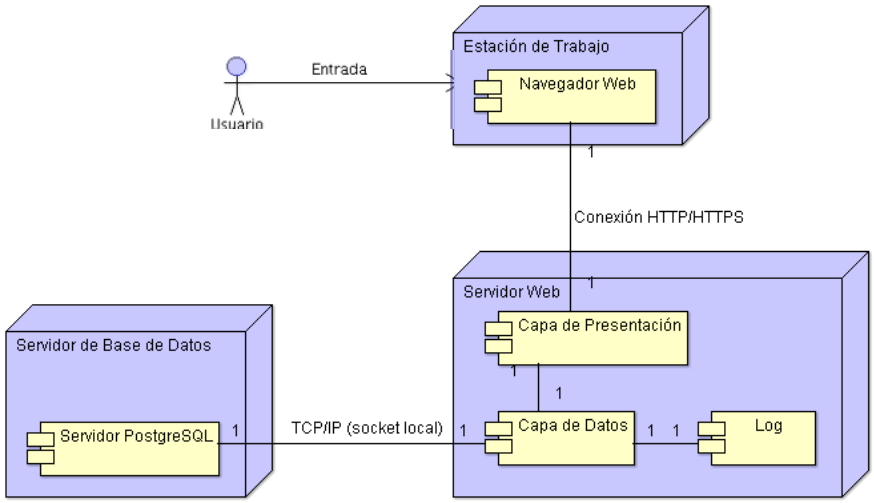


***Figura 39: Diagrama de Despliegue del Proyecto (*)***

### 4.6.4 Arquitectura de Software.

#### *4.6.4.1 Introducción.*

##### *4.6.4.1.1 Propósito.*

Este documento tiene como propósito describir la vista general de la arquitectura de software del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro, para lo cual se utilizan vistas arquitectónicas a fin de representar los diferentes aspectos del sistema.

##### *4.6.4.1.2 Alcance.*

Este documento de arquitectura de software proporciona una descripción del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro. Este documento ha sido generado directamente del análisis de los casos de uso a automatizar y del modelo de diseño.

#### *4.6.4.2 Representación arquitectónica.*

**Modelo de Caso de Uso:** Describe los procesos que brindarán al negocio la funcionalidad automatizada deseada y cómo funcionan internamente, contiene el modelo de caso de uso.

**Modelo de Análisis:** Describe un primer bosquejo de las clases de análisis que servirán de soporte para el diseño.

**Modelo de la Experiencia del Usuario:** Describe las pantallas del sistema, el contenido dinámico de pantallas y como el usuario navega a través de las pantallas para ejecutar las funcionalidades del sistema.

**Modelo de Diseño:** Describe las partes arquitectónicas significativas del modelo de diseño, tales como su descomposición en módulos y paquetes.

En el desarrollo del sistema se ha utilizado el patrón de arquitectura Modelo Vista Controlador (MVC).

#### *4.6.4.3 Objetivos arquitectónicos y coacciones.*

El principal objetivo del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro es poder realizar reservas por Internet, además se podrá obtener reportes actualizados de los procesos.

Todos los requerimientos estipulados en el documento de la Visión deben ser tomados en consideración durante el desarrollo de la arquitectura.

La construcción principal del diseño y de la implementación de la aplicación debe funcionar bajo una plataforma que consiste en los siguientes componentes:

**Lenguaje de programación:** Java.
**Entorno de Desarrollo:** NetBeans 7.2.1
**Servidor:** Windows Server 2003.
**Sistema de Gestión de Base de Datos:** Postgresql 9.0
**Servidor de Aplicaciones:** GlassFish 3.1
**Arquitectura Tecnológica:** JEE (Java Edición Empresarial) 6.

#### *4.6.4.4 Vista de casos de uso.*

Nos presenta una vista de los casos de uso de la arquitectura de software. La vista de casos de uso es la entrada importante de la selección de contexto y/o los casos de uso que son el punto de una interacción. Los casos de uso del sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro.

##### *4.6.4.4.1 Casos de uso arquitectónicamente significativos.*

Los caso de uso arquitectónicamente significativos son aquellos que representan las partes más críticas de la arquitectura del sistema y demuestran la funcionalidad del sistema. Para el sistema hotelero, los casos de uso significativos han sido priorizados en base al soporte que brindan a las metas del negocio.

**REGISTRAR NUEVA RESERVACIÓN**

Permitirá incrementar el número de reservas, ya que podrán ser realizadas directamente por el cliente desde cualquier lugar.

**REGISTRAR CLIENTES**

Permitirá a la empresa llevar el registro de sus clientes. Facilitando el análisis de su estado actual.

### *4.6.4.5 Vista lógica.*

Esta sección describe la estructura lógica del sistema. Empieza con la descripción de la arquitectura y después presenta sus elementos estructurales y del comportamiento dominantes.

#### *4.6.4.5.1 Elementos del modelo arquitecturalmente significativo.*

El sistema hotelero para la Empresa Pública Santa Agua de Chachimbiro, está compuesto por los siguientes módulos: Reservación, Recepción, Facturación y Seguridad.

Cada uno de los componentes del negocio se divide en las tres capas del patrón de arquitectura Modelo Vista Controlador (MVC).

1. Lógica de la presentación,
2. Lógica del negocio, y
3. Lógica de la integración.

Es decir, la arquitectura descompone los sistemas a la largo de dos dimensiones:

1. La primera dimensión, a lo largo de las líneas de la funcionalidad del sistema.
2. La segunda dimensión, en capas comúnmente reconocidas que separan tres clases de preocupaciones:
   a) Preocupaciones de la presentación, o cómo manejar la comunicación con el usuario y controlar su acceso a los servicios y a los recursos de sistema.
   b) Preocupaciones de negocio, o cómo organizar los elementos del sistema que realizan funciones de los servicios del negocio y del sistema, y
   c) Preocupaciones de la integración, o cómo conectar los elementos del sistema con el mecanismo de la persistencia, otros sistemas, dispositivos físicos, etc.

## 4.7 Pruebas

A continuación se muestran las especificaciones de casos de prueba funcionales de los casos de uso incluidos en el proyecto de desarrollo de software.

### 4.7.1 Especificación del caso de prueba: “Login - Logout”.

#### *4.7.1.1 Introducción.*

En la presente sección se detallan las pruebas a realizarse para los escenarios del caso de uso *“Login – Logout”.*

*4.7.1.1.1 Propósito.*

Verificar que el caso de uso *"Login – Logout"* esté correctamente implementado, y que se cumplan las especificaciones funcionales y no funcionales.

*4.7.1.1.2 Alcance.*

Sólo se prueban los escenarios mencionados en el documento de Especificación de Caso de Uso: *"Login – Logout".*

*4.7.1.1.3 Definiciones, acrónimos y abreviaciones.*

En la presente sección se deben tomar en cuenta los siguientes términos:

**Autenticar:** Se refiere al hecho de haber ingresado datos para que la aplicación pueda identificar que la persona que intenta acceder a su contenido es quien dice ser. A esta acción se lo denomina técnicamente Login.

**Sesión:** Período de tiempo de actividad que un usuario pasa en el sistema, desde que hace el Login hasta que hace Logout.

### *4.7.1.2 Escenarios de prueba.*

El presente documento contiene los escenarios detallados en la especificación del caso de uso *"Login – Logout"* y cada escenario tiene una breve descripción de lo que trata, el flujo de actividades que contiene los pasos a realizarse para cumplir el escenario y los puntos de control que indican los pasos donde evaluar exhaustivamente.

*4.7.1.2.1 Escenario: Flujo básico.*
Validar que el usuario pueda iniciar y terminar su sesión de manera correcta.

**Descripción:** Es el escenario ideal del caso de uso, no deberían existir errores.

**PRECONDICIONES**

El usuario no debe tener su sesión activa en el sistema. Se debe haber creado el usuario y sus datos indicando como dato de entrada.

**DATOS DE ENTRADA**

Se accederá al sistema con el usuario *"prueba"*[119] cuya contraseña es *"987"*. Su perfil es *Administrador.*

**FLUJO DE ACTIVIDADES**

| Paso | Instrucción | Resultado esperado |
|---|---|---|
| 1 | El usuario ingresa a la interfaz de inicio de sesión accediendo a la url de la aplicación. | El sistema muestra la interfaz de inicio de sesión o logeo en la que se solicita cuenta de usuario y contraseña. |
| 2 | El usuario ingresa a su cuenta de usuario y contraseña y selecciona la opción "Login". | El sistema muestra la interfaz a la que tiene acceso. |
| 3 | El usuario selecciona la opción "Cerrar Sesión" | El sistema solicita la confirmación de cierre de sesión. |

***Tabla 70: Flujo de actividades del escenario Flujo Básico del Caso de Prueba "Login-Logout" del Proyecto (*)***

**PUNTOS DE REVISIÓN**

| Paso | Punto de Control | Validación a realizar |
|---|---|---|
| 1 | Carga de la página de autentificación (login). | Al momento de intentar acceder al sistema, este revisa si la sesión del usuario existe o si está activa. Como no lo está, muestra la |

[119] Este usuario es utilizado solo para propósitos del documento. Luego de concluidas las pruebas, el usuario es borrado de la base de datos.

| | | |
|---|---|---|
| | | página de Login con los campos para el ingreso del usuario la contraseña. |
| 2 | Acceso del usuario al sistema. | El usuario ha iniciado sesión en el sistema, con el dato referente a la sesión. |
| 3 | El sistema muestra la interfaz de inicio de sesión. | El sistema debe mostrar la interfaz de logeo. |

***Tabla 71: Punto de revisión del escenario Flujo Básico del Caso de Prueba "Login-Logout" del Proyecto (*)***

*4.7.1.2.2 Escenario: Error de usuario.*

Verificar el reconocimiento de las cuentas de usuario validando su existencia en la base de datos.

**Descripción:** Comprobar que una cuenta se encuentra o no registrada en la base de datos.

**PRECONDICIONES**

La cuenta de usuario no ha sido registrada en la base de datos.

**DATOS DE ENTRADA**

Un usuario *"may"* con clave *"1523"*.

**FLUJO DE ACTIVIDADES**

| Paso | Punto de Control | Validación a realizar |
|---|---|---|
| 1 | El usuario ingresa su cuenta de usuario. | El sistema muestra mensaje de error "Usuario incorrecto" y retorna a la interfaz que lo solicitó. |

***Tabla 72: Flujo de actividades del escenario Error de usuario del Caso de Prueba "Login-Logout" del Proyecto (*)***

**PUNTO DE CONTROL**

Al concluir la acción, muestra el mensaje correspondiente

**PUNTO DE REVISIÓN**

| Paso | Punto de Control | Validación a realizar |
|---|---|---|
| 1 | Muestra mensaje de error. | Verificar que los datos sean los de la base de datos, comprobar que el sistema muestre el mensaje de acuerdo al error ocurrido. |

***Tabla 73: Punto de revisión del escenario Error de usuario del Caso de Prueba "Login-Logout" del Proyecto (*)***

*4.7.1.2.3 Escenario: Error de contraseña.*

Verificar el reconocimiento de la contraseña del usuario validando su existencia en la base de datos.

**Descripción:** Comprobar que una cuenta se encuentra o no registrada en la base de datos.

**PRECONDICIONES**

El usuario a utilizarse tiene una contraseña.

**DATOS DE ENTRADA**

Se intentará acceder al sistema con el usuario *"prueba"* cuya contraseña es *"987"*, sin embargo se utilizará la contraseña *"1523".*

## FLUJO DE ACTIVIDADES

| Paso | Punto de Control | Validación a realizar |
|---|---|---|
| 1 | El usuario ingresa su contraseña. | El sistema muestra mensaje de error "Contraseña inválida" y retorna a la interfaz que lo solicitó. |

***Tabla 74: Flujo de actividades del escenario Error de contraseña del Caso de Prueba "Login-Logout" del Proyecto (*)***

## PUNTO DE CONTROL

Al concluir la acción, muestra el mensaje correspondiente

## PUNTO DE REVISIÓN

| Paso | Punto de Control | Validación a realizar |
|---|---|---|
| 1 | Muestra mensaje de error. | Verificar que los datos sean los de la base de datos, comprobar que el sistema muestre el mensaje de acuerdo al error ocurrido. |

***Tabla 75: Punto de revisión del escenario Error de contraseña del Caso de Prueba "Login-Logout" del Proyecto (*)***

# CAPÍTULO V

# 5 Conclusiones y Recomendaciones

## 5.1 Conclusiones

1. El estudio de los patrones de diseño genera una gama de posibilidades de implementación para todo tipo de aplicaciones.

2. Un patrón de diseño es la recopilación de soluciones dadas para un problema encontrado durante el desarrollo.

3. El objetivo principal de los patrones de diseño es simplificar el desarrollo, dotando a las aplicaciones de escalabilidad, flexibilidad y reusabilidad.

4. El uso de patrones de diseño de JEE 6 en conjunto con las funcionalidades que esta plataforma ofrece, genera aplicaciones robustas con muy poco esfuerzo.

5. Existe mayor beneficio al desarrollar una aplicación distribuida en capas usando patrones de diseño de JEE, pues facilita el mantenimiento de la aplicación.

6. Un sistema basado en patrones de diseño y correctamente implementado, no precisará mayores modificaciones cuando se produzcan cambios en los requerimientos, ya que al crear un sistema utilizando patrones, la estructura básica del software es flexible y reutilizable pues se han tomado en cuenta, en la etapa del diseño, los posibles cambios que pueden surgir en el futuro para la aplicación.

7. Existe un gran número de patrones de diseño y estrategias para implementarlos, la elección dependerá de la complejidad del sistema y de las habilidades del desarrollador.

8. Los patrones de diseño de la plataforma JEE 6 evolucionaron, de manera que son más potentes y brindan mayores ventajas que los patrones de J2EE.

9. Aunque la funcionalidad de los patrones J2EE es mínima en comparación a los beneficios de los patrones de diseño JEE 6 y de la plataforma en sí, pueden ser utilizados todavía, esto dependerá del criterio del desarrollador.

10. El uso de los patrones DAO y Services simplificaron significativamente el desarrollo del sistema hotelero para la empresa Pública Santa Agua de Chachimbiro cuyo enfoque principal es la realización de reservas por Internet.

## 5.2 Recomendaciones

1. Continuar investigando, estudiando y aplicando los distintos patrones de diseño para obtener los beneficios derivados de estos.

2. Se recomienda la inclusión de patrones de diseño en el desarrollo de software por necesidad y para solucionar problemas presentes en el sistema en desarrollo; pero no agregarlos sin mayor conocimiento de los mismos.

3. No aplicar los patrones de diseño de forma indiscriminada, ya que esto puede complicar el diseño y la implementación de un sistema. Los patrones pretenden generar sistemas reutilizables, portables, escalables, con código legible, pero alcanzar estos objetivos puede generar mayores problemas de diseño e implementación, si no se aplican de la manera correcta.

4. Se recomienda leer la documentación del patrón, especialmente la sección *consecuencias*, porque dará una idea del resultado que se obtendrá al utilizar el patrón en un contexto determinado.

5. Las estrategias de implementación expuestas en este documento y que forman parte de la descripción original de cada patrón, no son una regla a seguir. Se recomienda su utilización si el desarrollo de la aplicación lo amerita.

6. Es recomendable, cuando se desea incluir un patrón de diseño en un sistema, recurrir a un desarrollador con experiencia en el manejo de patrones, pues su uso apropiado demanda tener cierta práctica al respecto.

7. Promover el desarrollo de aplicaciones empresariales y web utilizando los patrones de diseño.

8. Se recomienda el uso de la plataforma Java Enterprise Edition, por todas las características y funcionalidades que ofrece.

## 5.3 Posibles temas de tesis

- Estudio de patrones de diseño de la plataforma .NET.

- Estudio comparativo de patrones de software.

- Estudio comparativo de los patrones de diseño de las plataformas JEE, .NET y PHP.

- Evaluación del patrón de software adoptado por un determinado aplicativo.

## TRABAJOS CITADOS


- ✓ (LIB01) Alexander, C., Ishikawa, S., Silvertein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A Pattern Language: Towns/Builder/Construction.* New York: Oxford University Press.

- ✓ (LIB02) Beck, K., Cunningham, W., Inc, A. C., & Inc, T. (1987). *Using Pattern Languages for Object-Oriented Programs.* Orlando.

- ✓ (LIB03) Erich Gamma, R. H. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Massachusetts: Addison Wesley.

- ✓ (LIB04) Martin Fowler, D. R. (2002). *Patterns of Enterprise Application Architecture.* Massachusetts: Addison Wesley.

- ✓ (LIB05) Deepak Alur, J. C. (2003). *Core J2EE Patterns: Best Practices and Design Strategies.* California: Sun Microsystems, Prentice Hall.

- ✓ (LIB06) William Crawford, J. K. (2003). *J2EE Design Patterns* (1st Edition ed.). California: O' Reilly.

- ✓ (LIB07) Bien, A. (2009). *Real World Java EE Patterns – Rethinking Best Practices.* Kentucky.

- ✓ (LIB08) Bien, A. (2011). *Real World Java EE Night Hacks — Dissecting the Business Tier.*

- ✓ (LIB09) Óscar Belmonte, C. G. (2012). *Desarrollo de Proyectos Informáticos con Tecnología Java.* Universitat Jaume I.

- ✓ (LIB10) Riehle, D., & Züllighoven, H. (1996). *Understanding and using patterns in software development.* John Wiley & Sons, Inc.

- ✓ (LIB11) Gabriel, R. P. (1996). *Patterns of Software: Tales from the Software Community.* Oxford University Press, Inc.

- ✓ (LIB12) Coplien, J. (1996). *Software Patterns.* SIGS Books.

- ✓ (WWW01) Meyrowitz, Norman K. Obtenido de http://www.informatik.uni-trier.de/~ley/db/conf/oopsla/oopsla87.html

- ✓ (WWW02) Obtenido de http://es.wikipedia.org/wiki/ASP.NET

- ✓ (WWW03) Obtenido de http://es.wikipedia.org/wiki/Java_EE

- ✓ (WWW04) Obtenido de http://es.wikipedia.org/wiki/PHP
- ✓ (WWW05) Obtenido de http://es.wikipedia.org/wiki/Sun_Microsystems
- ✓ (WWW06) Obtenido de http://www.oracle.com/technetwork/java/patterns-139816.html
- ✓ (WWW07) Obtenido de http://es.wikipedia.org/wiki/Java_Community_Process
- ✓ (WWW08) Obtenido de http://es.wikipedia.org/wiki/SDK
- ✓ (WWW09) Obtenido de http://es.wikipedia.org/wiki/Java_EE
- ✓ (WWW10) Obtenido de http://www.java.com/es/download/faq/techinfo.xml
- ✓ (WWW11) Obtenido de http://www.oracle.com/technetwork/java/javaee/overview/index.html
- ✓ (WWW12) Obtenido de http://es.wikipedia.org/wiki/Plain_Old_Java_Object
- ✓ (WWW13) Obtenido de http://es.wikipedia.org/wiki/Java_Servlet
- ✓ (WWW14)OracleCorporation. (s.f.). *The Java EE 6 Tutorial.* Obtenido de http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf
- ✓ (WWW15) Miguel Abarca C., G. D. (2009). *Desarrollo Básico de Aplicaciones en la Plataforma J2EE.* Obtenido de http://xxito.files.wordpress.com/2008/05/manualj2ee.pdf
- ✓ (WWW16) Obtenido de http://es.wikipedia.org/wiki/Java_Archive
- ✓ (WWW17) Obtenido de http://es.wikipedia.org/wiki/WAR_(archivo)
- ✓ (WWW18) Obtenido de http://es.wikipedia.org/wiki/Enterprise_JavaBeans
- ✓ (WWW19) Obtenido de http://netbeans.org/images_www/visual-guidelines/NB-IDE-logo.png
- ✓ (WWW20) Obtenido de http://es.wikipedia.org/wiki/NetBeans
- ✓ (WWW21) Obtenido de http://netbeans-ide.softonic.com/
- ✓ (WWW22) Obtenido de http://www.postgresql.org/about/press/presskit92/en/#logos
- ✓ (WWW23)Quiñones, E. (s.f.). Obtenido de

- ✓ (WWW24)http://www.postgresql.org.pe/articles/introduccion_a_postgresql.pdf
- ✓ (WWW25) Obtenido de http://es.wikipedia.org/wiki/PostgreSQL
- ✓ (WWW26) Obtenido de http://es.wikipedia.org/wiki/GlassFish
- ✓ (WWW27) Obtenido de http://glassfish.softonic.com/
- ✓ (WWW28) Obtenido de http://hub.opensolaris.org/bin/view/Project+mx/licensing_faq
- ✓ (WWW29) Obtenido de http://www.gnu.org/licenses/licenses.es.html
- ✓ (WWW30) Obtenido de http://www.gnu.org/home.es.html
- ✓ (WWW31) Obtenido de http://es.wikipedia.org/wiki/JavaServer_Faces
- ✓ (WWW32) Obtenido de http://es.wikipedia.org/wiki/Proceso_Unificado_de_Rational
- ✓ (WWW33)http://www.programacion.com/articulo/catalogo_de_patrones_de_diseno_j2ee__i__capa_de_presentacion_240.
- ✓ (WWW34) Obtenido de http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o
- ✓ (WWW35) Obtenido de http://es.wikipedia.org/wiki/GRASP
- ✓ (WWW36)Obtenido de http://es.scribd.com/doc/76037307/13/Patron-de-diseno-seguro-%E2%80%9CGateKeeper%E2%80%9D
- ✓ (WWW37)Mejía, D. P. (s.f.). Recuperado el 4 de Noviembre de 2012, de http://delta.cs.cinvestav.mx/~pmejia/softeng/Patrones.ppt
- ✓ (WWW38) Obtenido de http://migranitodejava.blogspot.com/2011/05/patrones-de-diseno-de-gof.html
- ✓ (WWW39) Obtenido de http://es.wikipedia.org/wiki/Antipatr%C3%B3n_de_dise%C3%B1o
- ✓ (WWW40) Obtenido de http://cursoj2ee.blogspot.com/2009/02/controlador-frontal.html
- ✓ (WWW41) Obtenido de http://diccionariowebdigital.blogspot.com/2011/03/smalltalk-es-un-lenguaje-de.html

- ✓ (WWW42) Universidad de Alicante. (13 de 10 de 2011). *Servicio de Informática*. Recuperado el 21 de 09 de 2012, de http://si.ua.es/es/documentacion/asp-net-mvc-3/2-dia/razor/helpers.html

- ✓ (WWW43) Wikipedia. (29 de 09 de 2012). Obtenido de http://es.wikipedia.org/wiki/JavaBean

- ✓ (WWW44)Díaz, M. (2003). *MOISESDANIEL.COM*. Obtenido de http://www.moisesdaniel.com/es/wri/disaplicj2ee.html

- ✓ (WWW45)Sun Microsystems, Inc. (2002). *ORACLE*. Obtenido de http://www.oracle.com/technetwork/java/catalog-137601.html

- ✓ (WWW46)Wikipedia. (20 de 05 de 2012). Obtenido de http://es.wikipedia.org/wiki/Descriptor_de_Despliegue

- ✓ (WWW47) Obtenido de http://www.corej2eepatterns.com/Patterns2ndEd

- ✓ (WWW48)Torrijos, R. L. (s.f.). *Programación en Castellano.* Obtenido de http://www.programacion.net/articulo/catalogo_de_patrones_de_diseno_j2ee_i_-_capa_de_presentacion_240
- ✓ (WWW49) Gracia, L. M. (s.f.). *Un poco de Java.* Obtenido de http://unpocodejava.wordpress.com/2011/01/10/tecnicas-de-bloqueo-sobre-base-de-datos-bloqueo-pesimista-y-bloqueo-optimista/

- ✓ (WWW50) Obtenido de http://glassfish.java.net/glassfish_buttons/glassfish_logo_transparent.png

- ✓ (WWW51) Obtenido de http://hub.opensolaris.org/bin/view/Project+mx/licensing_faq


- ✓ (*) Fuente: Maricruz Acosta - Autora