Estudio de patrones de diseño en plataforma Java Enterprise Edition versión 6 para el desarrollo de aplicaciones web

Acosta, Maricruz[1]

[1]Universidad Técnica del Norte-UTN, Ibarra, Imbabura

jmary_86@hotmail.com

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.

Palabras Claves

Patrones, JEE6, Diseño, Aplicaciones, Web.

Abstract. *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.

Keywords

Patterns, JEE6, Design, Applications, Web.

1. Introducción

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.

Los patrones de diseño se originaron como conceptos de arquitectura civil por el arquitecto Christopher Alexander [1]. A continuación, se publica el libro *Core J2EE Patterns*[2], enfocado en las buenas prácticas y estrategias de diseño para aplicaciones empresariales. 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[3], 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[4], es publicado como

complemento al libro anterior, en él se refleja la utilización de los patrones en aplicaciones web.

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 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.

2. Marco Teórico

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.

Java Enterprise Edition existe desde el año 1998 como iniciativa de un grupo de ingenieros de Sun Microsystems, 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* 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.

2.1 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.

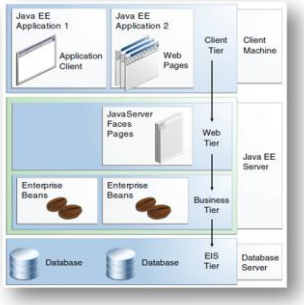

Figura. 1. Aplicaciones Multicapa[1]

2.2 Características y servicios de JEE

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. Cuenta con nuevas características como: perfiles, contextos e inyección de dependencia, nuevas características para EJBs, Servlets y para componentes JSF.

Entre los servicios que JEE provee y que se pueden emplear en cualquier servidor de aplicaciones que siga este estándar se encuentran: Java Transaction API (JTA), Java Persistente API (JPA), Java Message Service (JMS), JavaMail, Arquitectura Java EE Connector (JCA), Arquitectura Java EE Connector (JCA), entre otros.

2.3 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

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

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.

2.4 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; además, incrementan el vocabulario de diseño de los desarrolladores, ayudando a diseñar desde un mayor nivel de abstracción.

2.5 Qué son 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.

"Los patrones de diseño JEE contienen las mejores soluciones para ayudar a los desarrolladores a diseñar y construir aplicaciones en la plataforma JEE" (Bien, Real World Java EE Patterns – Rethinking Best Practices, 2009).

2.6 Diseño de aplicaciones web con patrones de diseño JEE

El diseño de aplicaciones Web basado en patrones JEE, se organiza intrínsecamente alrededor de la utilización de varios elementos: un controlador frontal, despachadores (Dispatchers), vistas compuestas, vistas estáticas (JSPs) y los ayudantes (helpers) de las vistas dinámicas formadas con JavaBeans. Esto puede relejarse en la figura 2, que ejemplifica un diseño basado en estos mecanismos.

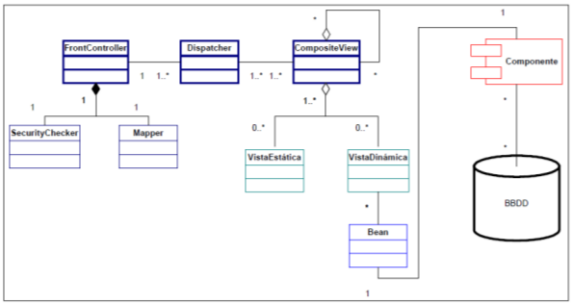

Figura. 2. Diseño basado en patrones JEE[2]

2.7 Catálogo de 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.

a) Patrones de la capa de presentación. Esta capa incluye Servlets, JSPs y otros elementos que generan interfaz de usuario del lado del servidor.

Los patrones de diseño de esta capa, 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. En este estudio se mencionan los siguientes patrones:

Intercepting Filter. Este patrón maneja varios tipos de peticiones que requieren un procesamiento determinado; es aplicado especialmente en procesos de validación de sesión. Se relaciona con los patrones Front Controller y Decorator.

Front Controller. El patrón Front Controller acepta todas las peticiones de un cliente y las direcciona a

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

los manejadores apropiados. Es relacionado con los patrones View Helper, Intercepting Filter, Service to Worker y Dispatcher View.

View Helper. Encargado de encapsular la lógica de acceso a bases de datos. Se relaciona con los patrones Business Delegate, Front Controller, Service to Worker y Dispatcher View.

Composite View. Utilizado cuando se requieren subvistas comunes, como encabezados y pies de página, en múltiples vistas, que pueden aparecer en diferentes lugares dentro de cada diseño de página. Se relaciona con el patrón View Helper.

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.

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

b) Patrones de la capa de negocio. Esta capa 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. En este estudio se mencionan los siguientes patrones:

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. Se relaciona con los patrones: Application Service, Session Facade, Gateway, Service, Data Access Object, Data Transfer Object.

Service. Es el nuevo nombre del patrón 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. Un Service es coordinado por Service Facades o Gateways, invoca DAOs y es capaz de producir DTOs.

Persistent Domain Object. 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. Se relaciona con los patrones de diseño Composite Entity, Domain Store, Business Object, Data Transfer Object.

Gateway. Un Gateway proporciona un punto de entrada a la raíz de los PDOs. Se relaciona con los patrones: Composite Entity, Domain Store, Business Object, Data Transfer Object, Service Facade, Service.

Fluid Logic. Los algoritmos que cambian a menudo requieren re-compilación e incluso redespliegue de toda la aplicación. El patrón Fluid Logic es un Service específico.

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.

Service Locator. Fue un patrón obligatorio en aplicaciones J2EE. Las principales razones para su retiro son:

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. 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.

Composite Entity. Representa un esquema de objetos. Las principales razones para su retiro son:

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. El patrón Composite Entity en JEE fue degradado a patrón de diseño Composite del catálogo GoF.

Value Object Assembler. Fue un patrón dedicado a fusionar, transformar o extraer datos desde diferentes fuentes de datos. Las principales razones para su retiro son:

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.

Business Delegate. Se utilizó para ocultar detalles de implementación del negocio y para reducir el acoplamiento entre éste y la capa de presentación. Las principales razones para su retiro son:

Ya no es necesario utilizar este patrón para separar la lógica de negocio de la presentación, porque las interfaces de negocio pueden ser inyectadas directamente en la mayoría de los componentes de presentación.

Con EJB 3.0 la interfaz de negocio es idéntica a la interfaz externa del Business Delegate. El Business Delegate hacía uso del patrón 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.

Domain Store. Este patrón fue utilizado para persistir de forma transparente un modelo de objetos siendo independiente del modelo. Su retiro se debe a que en JEE el Entity Manager es considerado un patrón Domain Store.

Value List Handler. Este patrón fue utilizado para proporcionar iteración entre los datos y el cliente. Las razones para su retiro son:

Desde la introducción de JPA, las entidades se pueden separar fácilmente sin esfuerzo adicional. La implementación del Value List Handler no es relevante, porque es la aplicación del patrón Iterator[3].

El Value List Handler se convirtió en una estrategia del patrón Paginator.

c) Patrones de la capa de integración. Esta capa agrupa componentes de conexión con otros sistemas, conectores con servicios, servicios de mensajería y otros. En esta capa se numeran los siguientes patrones:

Data Access Object. Es utilizado para separar la lógica de negocio del acceso a datos. En un entorno Java EE 6, se reduce 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. Debe ser invocado desde un componente de la lógica del negocio y no directamente desde la presentación.

El uso de un DAO dedicado es opcional, puede ser reemplazado con un EntityManager inyectado en un Service. Un DAO es usado por un Service o Service Facade y podrá utilizar JCA Genérico para acceder a recursos propietarios heredados.

Transfer Object y Data Transfer Object. La responsabilidad principal de estos patrones es la optimización de la transferencia de datos. Los patrones Data Access Object, Service Facade y Service pueden consumir y producir DTOs.

En J2EE, su uso 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; entonces, cada cambio accionado por una solicitud llega a las entidades JPA.

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.

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.

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. Es un conjunto de interfaces API y SPI orientado a la conexión.

Se compone de dos interfaces, cuatro clases, y un archivo XML. Es completamente transparente para el desarrollador. Los conectores pueden ser accedidos por los patrones Service y Service Facade.

Asynchronous Resource Integrator. 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.

Este patrón puede trabajar directamente con DAOs y los Servicios pueden invocarlo mediante el envío de mensajes JMS.

d) 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:

Service Starter. Su uso debe satisfacer las siguientes necesidades: Se necesita una forma portable para iniciar los servicios con impaciencia y 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.

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

Invoca Service, Service Facade o Gateway para producir su inicialización, es técnicamente idéntico al patrón Singleton.

Singleton. Su responsabilidad es proporcionar estado compartido y administración de concurrencia. Puede acceder a Service Facades existentes, servicios, y recursos para buscar información necesaria y almacenarla en caché posteriormente.

Bean Locator. El patrón Bean Locator es responsable de localizar EJBs y de la construcción de los nombres JNDI globales. Retorna beans remotos o locales con estado, sin estado y Singleton, y, sólo debe utilizarse en clases donde la Inyección de Dependencia no funciona o no está disponible.

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 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.

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. La funcionalidad de control adicional debe ser fácil de remover antes del despliegue en producción.

Se puede conectar a una clase determinada empleando anotaciones o XML. 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.

Dependency Injection Extender. El Extensor de Inyección de Dependencia (DIE) es práctico para la reutilización de fragmentos pequeños de la funcionalidad existente.

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.

Payload Estractor. 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.

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

Resource Binder. La carpeta de recursos 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.

Los recursos expuestos por este patrón son consumidos en beans de sesión y por lo tanto en los patrones Services, Service Facades y DAOs.

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.

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.

2.8 Aplicativo

Con el fin de aplicar lo aprendido en el transcurso del estudio, se planteó el desarrollo de un pequeño sistema hotelero utilizando como herramientas de desarrollo el IDE NetBeans versión 7.2.1, Postgresql 9.0 y el servidor de aplicaciones GlassFish 3.1; y, para la documentación metodología RUP.

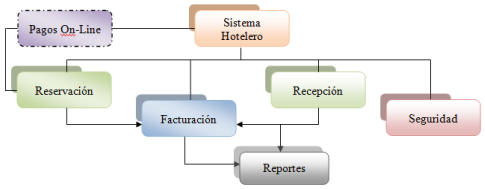

Figura. 3. Módulos del sistema hotelero[4]

Módulo Reservación. Este módulo permite: Controlar la 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. Este módulo permite: Realizar el registro de huéspedes con o sin reservación. Modificación de datos de reserva y recepción. Consulta de disponibilidad de cabañas. Consultas de reservaciones.

Módulo Facturación. Este módulo se encarga de emitir el respectivo comprobante de pago.

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

[4] Fuente: Autora

Módulo Reportes. Permite generar el listado de reservaciones y ocupación de habitaciones.

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

Figura. 4. *Interfaz de Usuario: Login del Proyecto*[5]

3. Resultados

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.

La plataforma JEE ofrece distintas posibilidades de diseño. Sin embargo, la utilización del patrón arquitectónico MVC y los Patrones de Diseño JEE 6 dan a las aplicaciones escalabilidad, facilitando futuras ampliaciones, migraciones y cambios en general. Esto se pudo comprobar en el transcurso del desarrollo del aplicativo donde fueron utilizados principalmente los patrones Data Access Object, Services y Paginator.

4. Conclusiones

El estudio de los patrones de diseño genera una gama de posibilidades de implementación para todo tipo de aplicaciones.

Un patrón de diseño es la recopilación de soluciones dadas para un problema encontrado durante el desarrollo.

El objetivo principal de los patrones de diseño es simplificar el desarrollo, dotando a las aplicaciones de escalabilidad, flexibilidad y reusabilidad.

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.

Existe mayor beneficio al desarrollar una aplicación distribuida en capas usando patrones de diseño de JEE, pues facilita el mantenimiento de la aplicación.

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.

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.

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.

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.

5. Recomendaciones

Continuar investigando, estudiando y aplicando los distintos patrones de diseño para obtener los beneficios derivados de estos.

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.

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.

Se recomienda leer la documentación del patrón, especialmente la sección *consecuencias*, porque dará una

[5] Fuente: Autora

idea del resultado que se obtendrá al utilizar el patrón en un contexto determinado.

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.

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.

Promover el desarrollo de aplicaciones empresariales y web utilizando los patrones de diseño.

Se recomienda el uso de la plataforma Java Enterprise Edition, por todas las características y funcionalidades que ofrece.

Agradecimientos

Quiero hacer extensivo un sincero agradecimiento a la Universidad Técnica del Norte en especial al personal Docente de la Carrera de Ingeniería es Sistemas Computacionales representados por los Ingenieros Jorge Caraguay, Mauricio Rea y José Luis Rodríguez, Director de Tesis, quienes con sus consejos y conocimientos hicieron posible la culminación de mi trabajo.

Referencias Bibliográficas

[1] 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.

[2] Deepak Alur, J. C. (2003). *Core J2EE Patterns: Best Practices and Design Strategies.* California: Sun Microsystems, Prentice Hall.

[3] Bien, A. (2009). *Real World Java EE Patterns – Rethinking Best Practices.* Kentucky.

[4] Bien, A. (2011). *Real World Java EE Night Hacks — Dissecting the Business Tier.*

[5] Óscar Belmonte, C. G. (2012). Desarrollo de Proyectos Informáticos con Tecnología Java. Universitat Jaume I.

[6] 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

[7] OracleCorporation. (s.f.). *The Java EE 6 Tutorial.* Obtenido de http://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf

[8] 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/

[9] Torrijos, R. L. (s.f.). *Programación en Castellano.* Obtenido de http://www.programacion.net/articulo/catalogo_de_patrones_de_dise_j2ee_i_-_capa_de_presentacion_240

[10] Sun Microsystems, Inc. (2002). *ORACLE.* Obtenido de http://www.oracle.com/technetwork/java/catalog-137601.html

[11] 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

[12] William Crawford, J. K. (2003). *J2EE Design Patterns* (1st Edition ed.). California: O' Reilly.

[13] Beck, K., Cunningham, W., Inc, A. C., & Inc, T. (1987). *Using Pattern Languages for Object-Oriented Programs.* Orlando.

[14] Erich Gamma, R. H. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Massachusetts: Addison Wesley.

[15] Martin Fowler, D. R. (2002). *Patterns of Enterprise Application Architecture.* Massachusetts: Addison Wesley.

[16] Riehle, D., & Züllighoven, H. (1996). *Understanding and using patterns in software development.* John Wiley & Sons, Inc.

[17] Gabriel, R. P. (1996). *Patterns of Software: Tales from the Software Community.* Oxford University Press, Inc.

[18] Coplien, J. (1996). *Software Patterns.* SIGS Books.

[19] Díaz, M. (2003). *MOISESDANIEL.COM.* http://www.moisesdaniel.com/es/wri/disaplicj2ee.html

[20] Mejía, D. P. (s.f.). Recuperado el 4 de Noviembre de 2012, de http://delta.cs.cinvestav.mx/~pmejia/softeng/Patrones.ppt

Fuente de consulta Wikipedia

[21] http://es.wikipedia.org/wiki/ASP.NET

[22] http://es.wikipedia.org/wiki/Java_EE

[23] http://es.wikipedia.org/wiki/PHP

[24] http://es.wikipedia.org/wiki/Sun_Microsystems

[25] http://es.wikipedia.org/wiki/Java_Community_Process

[26] http://es.wikipedia.org/wiki/SDK

[27] http://es.wikipedia.org/wiki/JavaBean

[28] http://es.wikipedia.org/wiki/Plain_Old_Java_Object

[29] http://es.wikipedia.org/wiki/Java_Servlet

[30] http://es.wikipedia.org/wiki/Java_Archive

[31] http://es.wikipedia.org/wiki/WAR_(archivo)

[32] http://es.wikipedia.org/wiki/Enterprise_JavaBeans

Otras fuentes

[33] http://www.programacion.com/articulo/catalogo_de_patrones_de_diseno_j2ee__i__capa_de_presentacion_240.

[34] http://cursoj2ee.blogspot.com/2009/02/controlador-frontal.html

[35] http://www.corej2eepatterns.com/Patterns2ndEd

[36] http://migranitodejava.blogspot.com/2011/05/patrones-de-diseno-de-gof.html

Sobre el Autor

Autora-Maricruz ACOSTA Egresada de la Carrera de Ingeniería en Sistemas Computacionales de la Universidad Técnica del Norte.