ARQUITECTURA POR COMPONENTES JEE, UN CASO PRÁCTICO

JEE COMPONENT ARCHITECTURE, A CASE STUDY

AUTOR
ALBA CONSUELO NIETO LEMUS
Msc. Ingeniería de Sistemas y Computación
*Universidad Distrital Francisco José de Caldas
Docente
Facultad de Ingeniería
acnietol@udistrital.edu.co
COLOMBIA

***INSTITUCIÓN**
Universidad Distrital Francisco José de Caldas
UDFJC
Institución Pública
Carrera 7 No. 40B – 53, Bogotá D.C.
ingsistemas@udistrital.edu.co
COLOMBIA

INFORMACIÓN DE LA INVESTIGACIÓN O DEL PROYECTO: Resultados obtenidos en la implementación de un prototipo de software para gestionar comisiones de estudio y permisos académicos concedidos a los docentes de la Universidad Distrital Francisco José de Caldas. El proyecto tuvo por objetivo mostrar cómo integrar frameworks como Seam, EJB, Hibernate y RichFaces con la arquitectura por componentes JEE para facilitar el proceso de desarrollo y promover la reutilización, escalabilidad y mantenimiento de una aplicación de software.

RECEPCIÓN: Agosto 15 de 2014 **ACEPTACIÓN:** Octubre 28 de 2014

TEMÁTICA: Ingeniería de Software

TIPO DE ARTÍCULO: Artículo de Investigación Científica e Innovación

Forma de citar: Nieto Lemus, A. C. (2015). Arquitectura por componentes JEE, un caso práctico. En R, Llamosa Villalba (Ed.). Revista Gerencia Tecnológica Informática, 14(38), 31-41. ISSN 1657-8236.

RESUMEN ANALÍTICO

Hoy en día el desarrollo de software está enfocado al manejo de requerimientos cambiantes. El proceso de desarrollo debe permitir, no solo la adaptación al cambio, sino también la reducción del esfuerzo, el tiempo y los costos, a la vez que debe promover la escalabilidad, fiabilidad y reutilización del software. Es por esto que el desarrollo de software basado en componentes surge como una línea de la ingeniería del software que construye y utiliza técnicas para la implementación de sistemas abiertos y distribuidos mediante el ensamblaje de partes reutilizables[17]. En este artículo se describen los principios de la arquitectura por componentes *Java Enterprise Edition* (*JEE*) y se presenta un caso práctico de implementación utilizando *Enterprise Java Beans* (*EJB's*), *Richfaces*, *Hibernate* y *Seam* en su construcción.

PALABRAS CLAVES: Arquitectura de software, Componente de software, Java, EJB, Hibernate, JEE, Richfaces, Seam, Stateless, Stateful, biyección, Facelets, Jboss.

ANALYTICAL SUMMARY

The software development is focused today on the management of changing requirements. The development process should allow not only adapt to change but also to the reduction of effort, time and costs, as well as to the promote of the scalability, the reliability and software reuse. That is why the development of component-based software emerges as a line of software engineering that builds and uses techniques for the implementation of open and distributed systems by assembling reusable parts [17]. This article describes the principles of the *Java Enterprise Edition* (*JEE*) component architecture and presents a practical implementation using Enterprise Java Beans (EJB's), Richfaces, Hibernate and Seam in its construction.

KEYWORDS: Software architecture, Software component, Java, EJB, Hibernate, JEE, Richfaces, Seam, Stateless, Stateful, biyeccion, Facelets, Jboss

INTRODUCCIÓN

Si bien no hay una única definición de arquitectura de software, la mayoría de ellas coinciden en asociarla con los componentes del sistema y sus interrelaciones. Según Rosen, Lublinsky, Smith y Balcer [12], una arquitectura de software es la descripción de un sistema de software en términos de sus principales componentes, sus relaciones y la información que pasa entre ellos. Se justifica particularmente en la medida en que las aplicaciones comienzan a ser más grandes, más integradas y deben ser implementadas usando una amplia variedad de tecnologías y disciplinas que necesitan ser orquestadas para asegurar la calidad del producto [2].

Más allá de describir componentes e interrelaciones, la arquitectura de software también tiene otros propósitos: sirve de mecanismo de comunicación entre los interesados en el proyecto integrando sus diferentes puntos de vista, promueve el re-uso al aplicarla a otros sistemas con características similares [2] y permite la co-evolución del sistema, necesaria porque los cambios en los requerimientos de negocio presionan la evolución del software al tiempo que los cambios tecnológicos presionan la evolución del negocio [10].

El desarrollo de software es un proceso no trivial que debe estar soportado en una estructura arquitectónica adecuada. Surgen así diferentes propuestas arquitectónicas como el modelo cliente-servidor centralizado o distribuido [13], la arquitectura de *n*-capas [5] [12] [15], la arquitectura por componentes [3] [16] y la Arquitectura Orientada a Servicios - SOA [12], entre otras. En este artículo se presenta un caso práctico de una arquitectura por componentes en *n*-capas (*Java Enterprise Edition - JEE*) como un estilo arquitectónico que promueve la flexibilidad, la facilidad

de desarrollo, la reutilización y la escalabilidad a la vez que considera requerimientos no funcionales dentro del proceso de desarrollo de software.

1. DESARROLLO DE SOFTWARE BASADO EN COMPONENTES: ANTECEDENTES

Los avances en la programación paralela y concurrente, junto con los requerimientos de integración de información localizada en distintos sitios geográficos dieron lugar al desarrollo de la tecnología para la construcción de aplicaciones distribuidas [17]. Esto sumado al auge de Internet potenció el desarrollo de sistemas descentralizados, abiertos y distribuidos.

La necesidad de dividir el sistema de software en partes relacionadas y residentes en distintas máquinas, junto con el principio de reutilización de software existente para construir aplicaciones empresariales, dio como resultado un nuevo paradigma de programación: el desarrollo de software basado en componentes, que propone metodologías y tecnologías para la construcción de aplicaciones mediante el ensamblaje de piezas de software reutilizables.

1.1 ¿QUÉ ES UN COMPONENTE DE SOFTWARE?

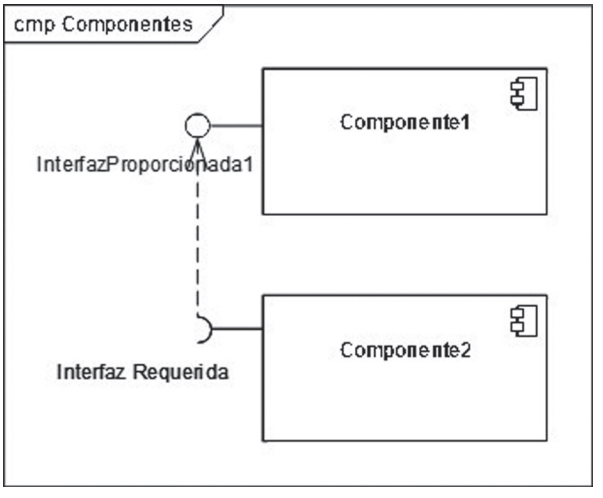
Aunque existen diversas definiciones, la dada por Szyperski [16] recoge los aspectos fundamentales de lo que es un componente de software:

"Un componente es una unidad binaria de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio".

Un componente es una unidad de software funcional autocontenida que es "empaquetada" junto con sus archivos y clases relacionadas para que pueda ser desplegado y utilizado por otros componentes. Dependiendo de la granularidad o nivel de detalle, un componente software puede ser desde una función de una librería matemática, un *Java Bean*[1], o una aplicación que puede ser usada por otra aplicación mediante una interfaz especificada [16]. Un componente puede presentarse en forma de código fuente o código objeto; puede estar escrito en un lenguaje funcional, procedimental o en un lenguaje orientado a objetos y puede ser tan simple como un botón de una *Interfaz Gráfica de Usuario* (*GUI*) o tan complejo como un subsistema. Un sistema es construido mediante el ensamblaje de componentes que exponen operaciones a través de interfaces y que a su vez consumen operaciones proporcionadas por otros componentes (ver figura 1).

Un componente de software tiene un ciclo de vida. Según Cheesman y Daniels [3] se identifican 5 diferentes "visiones" en las que el término componente aparece en las etapas de desarrollo de software:

FIGURA 1. Componente de software.



- Especificación del componente: Define la unidad de software que describe el comportamiento (interfaces) del componente y la unidad de implementación. La implementación realiza la especificación.
- Interfaz del componente: Define el conjunto de operaciones ofrecidas por un objeto componente.
- Implementación del componente: Realización de la especificación del componente que puede ser implantada, instalada y reemplazada de forma independiente en uno o más archivos.
- Componente instalado: Copia instalada de una implementación del componente.
- Objeto componente: Instancia de un "componente instalado". Es un objeto con su propio estado e identidad que ejecuta el comportamiento implementado. Un componente instalado puede tener múltiples objetos componente o uno solo.

Existen modelos de software basados en componentes, como EJB [5] o CCM [11], que no soportan completamente el ciclo de vida del componente, pues se enfocan sólo en la implementación y en la instalación, pero no en el diseño. Koziolek, Becker et al. [8] proponen un modelo incremental para definir los componentes en las distintas etapas del diseño, combinando especificaciones

1 Los *JavaBeans* son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java. Se usan para encapsular varios objetos en un único objeto (la vaina o Bean en inglés), para hacer uso de un solo objeto en lugar de varios más simples [4].

gruesas de nuevos componentes con especificaciones detalladas de componentes ya existentes, pasando por tres etapas: especificación de las interfaces provistas, refinamiento del componente con las interfaces necesarias para proveer la funcionalidad e implementación del componente ensamblando otros componentes o desarrollándolo completamente.

2. ARQUITECTURA MULTICAPAS

Una arquitectura multicapas promueve la separación de responsabilidades por capas. En el caso particular de 3-capas se separa la presentación de la lógica de negocio y ésta de la de datos. La capa de presentación no accede directamente a la base de datos, sino que lo hace únicamente a través de la capa de negocio. La arquitectura multicapas introduce muchas mejoras importantes dentro del diseño de la aplicación, incluyendo la flexibilidad a través de una adecuada separación entre la capa de presentación y la lógica de negocio [12]:

- La presentación puede cambiar sin impactar la lógica existente. Además, la misma lógica de negocio puede ser reutilizada en diferentes tipos de presentación.
- Con la introducción de una capa de negocio separada, se simplifica la integración de múltiples fuentes de datos dentro de una aplicación.
- Permite configurar las opciones de despliegue.

La arquitectura *n*-capas es una generalización de la arquitectura 3-capas, en la que se incluye una capa adicional, responsable de la interacción con el usuario, y capas adicionales para manejar la conexión a la fuente de datos. En la figura 2 se ilustran las capas típicas de una arquitectura multicapas:

Capa Cliente: Es la responsable de la interacción entre el sistema y el usuario a través de una presentación específica. La separación con la capa de negocio permite manejar diferentes tipos de interfaz de usuario, como un browser web o un dispositivo móvil.

FIGURA 2. Capas típicas de una arquitectura multicapas.

Capa de presentación: Permite al sistema soportar múltiples interacciones con un único usuario. Coordina y mantiene una sesión de usuario, manipulando los datos asociados con la sesión y con la interacción con la capa de negocio; coordina y mantiene la integridad con múltiples actividades para el mismo usuario; ejecuta procesos que no requieren acceso a los recursos empresariales. Es de aclarar que no hay una instancia de los recursos de negocio para cada usuario, sino que hay sólo una instancia de la capa de negocio y de datos que es compartida por todos los usuarios del sistema.

Capa de negocio: Es responsable de implementar los procesos y las entidades de negocio, y de hacer sus funciones disponibles a través de interfaces. Mantiene la integridad y fuerza el cumplimiento de reglas de negocio; maneja el control de transacciones. Provee un único punto de acceso a los recursos empresariales de manera que estos pueden ser compartidos y reutilizados por múltiples aplicaciones y usuarios.

Capa de integración: Es la responsable del manejo y acceso a los recursos empresariales compartidos. Provee acceso a las fuentes de datos y sistema legados[2].

Capa de datos: Fuentes de datos con cualquier modelo (sistemas de bases de datos o archivos de datos) y sistemas legados.

3. ARQUITECTURA JEE

En el mundo de la tecnología de la información, las aplicaciones empresariales deben ser diseñadas, construidas y producidas con menos dinero, con mayor rapidez y con menos recursos [5]. Además de estas restricciones, las aplicaciones deben integrar sistemas heterogéneos y distribuidos, y considerar el escalamiento y la reutilización como parte de la solución. Desarrollar este tipo de aplicaciones requiere bastante esfuerzo porque se deben escribir muchas líneas de código para el manejo transaccional, la concurrencia, la asignación de recursos, y otra serie de actividades complejas de bajo nivel de detalle. Estas condiciones han llevado a distribuir responsabilidades desde el punto de vista técnico y de desarrollo, delegando tareas al servidor de la aplicación, de tal modo que el desarrollador se pueda centrar en la implementación de la lógica del negocio.

La plataforma JEE [5], construida por el Java Community Process JCP, propone un modelo arquitectónico distribuido, multicapas, basado en componentes

2 Un sistema heredado (o sistema legacy) es un sistema informático que ha quedado anticuado pero continúa siendo utilizado por el usuario (típicamente una organización o empresa) y no se quiere o no se puede reemplazar o actualizar de forma sencilla [14].

reutilizables que pueden ser instalados en distintas máquinas, con un esquema de seguridad unificado, un control flexible de transacciones e integración de servicios web basados en XML (Extensible Markup Language). Está basada en protocolos y estándares abiertos para diseñar, desarrollar, ensamblar y desplegar aplicaciones empresariales.

Los componentes son escritos en lenguaje Java y compilados como cualquier otro programa; la diferencia entre el componente JEE y una clase Java estándar es que los componentes JEE son ensamblados dentro de aplicaciones JEE, verificados para asegurar que estén bien formados y que sean compatibles con la especificación JEE y desplegados en producción dentro de un servidor de aplicaciones JEE.

3.1 CAPAS DE LA ARQUITECTURA JEE [5]

La figura 3 ilustra las capas y componentes de la arquitectura JEE:

Capa cliente: Puede estar compuesta de un cliente web o una aplicación cliente.

- *Cliente web*: Es un "cliente delgado" que interactúa con el usuario; usualmente no accede directamente a la fuente de datos ni ejecuta lógica compleja, sino que accede mediante un *browser* a los componentes de negocio que se encuentran del lado del servidor de la aplicación. Con un cliente delgado, la responsabilidad de la seguridad, desempeño y confiabilidad se le delega a tecnologías del lado del servidor JEE.

- *Aplicación cliente*: Provee una interface GUI escrita en *AWT* o *Swing*. Puede acceder a la capa web del servidor comunicándose con un servlet a través de HTTP.

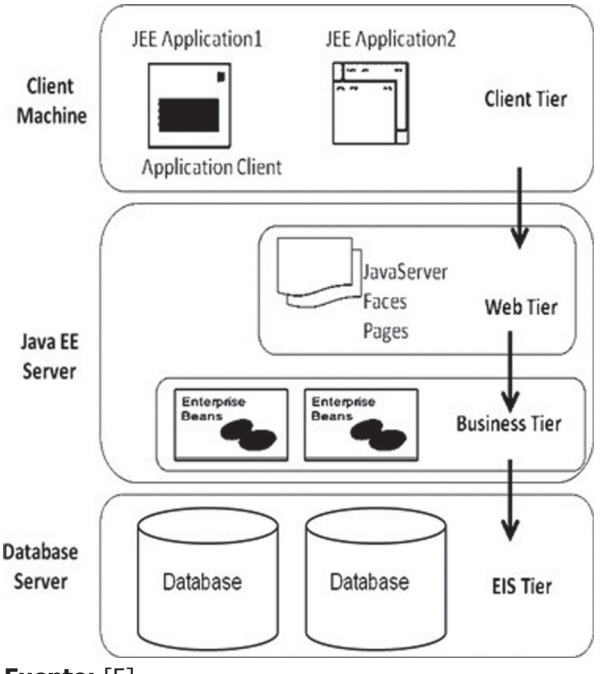
Capa web: Es la capa de presentación. Puede incluir

- *Servlets*: Son clases java que procesan dinámicamente solicitudes (*requests*) y construyen respuestas (*responses*).

- *Páginas JSP*: Son documentos basados en texto que se ejecutan como los *servlets* pero procesan de una manera más natural el contenido estático.

- *Java Server Faces*: Es una tecnología que se construye sobre servlets o JSP y que posee un conjunto de componentes GUI para aplicaciones web.

FIGURA 3. Componentes de la Arquitectura JEE.



Fuente: [5].

Capa de negocio: Maneja la lógica de negocio a través de los componentes *Enterprise Java Bean EJB*. Un componente EJB puede ser de uno de los siguientes tipos:

- *Session bean*: Representa una conversación trasiente o persistente con el cliente. Encapsula la lógica del negocio que puede ser invocada programáticamente por un cliente local, remoto o un servicio web. Cuando el cliente termina la ejecución, el *Session bean* y sus datos son descartados.
- Los *Session bean* pueden ser de tipo ***Stateful*** (mantiene el estado conversacional con el cliente, es no compartido, cada instancia está asociada a la sesión de un cliente), ***Stateless*** (no mantiene un estado conversacional con el cliente, es compartido: excepto durante la invocación de un método, todas las instancias del bean son equivalentes, permitiendo que el servidor asigne la instancia a otro cliente) y ***Singleton Session Bean*** (se instancia uno por aplicación y existe para el ciclo de vida de la aplicación; una sola instancia del bean es compartida y accedida concurrentemente por varios los clientes).
- *Entity bean*: Representa datos persistentes en una base de datos. Si la sesión termina o el servidor falla, los servicios subyacentes aseguran que el *entity bean* sea guardado.

- *Message bean*: Combina funciones de un *Session bean* y Java Masaje *Services* (*JMS*) permitiendo que los componentes del negocio reciban mensajes asincrónicamente.

Capa de conectividad y de recursos: Compuesta por el software e infraestructura para manejar persistencia. Incluye sistemas manejadores de bases de datos relacionales (*RDBMS*) como *Oracle*, *Mysql*, *SysBase* o cualquier otro, sistemas legados (Cobol) o sistemas de archivos (Excel).

3.2 SERVIDOR JEE

Provee servicios subyacentes y transversales a una aplicación como seguridad, transaccionalidad, servicio de nombres y conectividad remota (acceso a recursos remotos como si estuvieran en la misma máquina virtual). También maneja servicios no configurables como el ciclo de vida del componente, el pool de conexiones con la base de datos y diferentes opciones de persistencia.

El servidor JEE provee un contenedor web para el manejo de la ejecución de las páginas web y servlets, y un contenedor para la ejecución de los *EJB's*. El proceso de despliegue de la aplicación se encarga de instalar cada componente dentro de su contenedor. Algunos ejemplos de servidores JEE son *Java System Application Server*, *Oracle Application Server*, *Jboss*, *GlassFish* y *WebLogic Server*.

4. FRAMEWORKS DE DESARROLLO QUE SE INTEGRAN CON LA ARQUITECTURA JEE

Con el fin de facilitar la construcción de aplicaciones por componentes, se han desarrollado algunos frameworks compatibles con la especificación JEE. En esta sección se incluyen los utilizados en la implementación del caso de estudio.

Java Server Faces – JSF [5]

Es un framework de la capa web basado en el patrón *MVC* (*Modelo Vista Controlador*), que utiliza servlets o páginas *JSP* y que incluye un conjunto rico de componentes GUI de alto nivel que encapsulan elementos HTML. Utiliza clases java denominadas *Backing Beans* para procesar eventos e invocar los métodos de los componentes EJB de negocio. *JSF* es un framework de presentación robusto que permite manejar características adicionales como reglas de navegación entre las páginas, validadores en los componentes GUI y soporte a la internacionalización.

JSF tiene diferentes implementaciones dentro del software libre. Una de ellas es *Richfaces*, que además de las características anteriores, soporta funcionalidades *Ajax* (*Asynchronous JavaScript And XML*) para mantener una comunicación asíncrona con el servidor de aplicaciones de manera que se pueden hacer cambios en las páginas sin necesidad de recargarlas completamente, mejorando así el desempeño de la aplicación.

Hibernate [6]

Es un mapeador objeto-relacional de software libre que permite la traducción de datos del modelo relacional al modelo de objetos: los datos de una base de datos relacional se deben convertir a objetos java (y viceversa) para manipularlos en la capa de negocio. Desde el punto de vista de la arquitectura JEE, el mapeador objeto-relacional integra la capa de datos con la capa de negocio.

Hibernate no solo se ocupa de la traducción objeto-relacional, sino que también ofrece facilidades de consulta y gestión de los datos, ocupándose del manejo de la conexión y administración de transacciones, liberando al desarrollador del 95% de las tareas comunes de programación relacionadas con el manejo de la persistencia eliminando la necesidad de procesar los datos usando *SQL* y *JDBC*. Además, *Hibernate* provee una capa de transparencia entre la capa de negocio y la de datos al encapsular las tareas de conexión y particularidades de implementación de cada vendedor de bases de datos relacionales.

Seam [7]

Es un framework de desarrollo de software libre para construir aplicaciones de internet en Java. Integra tecnologías como *AJAX* y *XML*, *JavaServer Faces*, *Java Persistence* e *Enterprise Java Beans*, entre otras. El objetivo de *Seam* es eliminar la complejidad de integración entre todas las capas de la arquitectura JEE, desde la capa de datos a la capa de presentación. Habilita a los desarrolladores para ensamblar aplicaciones web complejas usando anotaciones simples en las clases Java, junto con un conjunto rico de componentes de GUI y muy poco código escrito en XML.

Entre las características más relevantes de *Seam* se encuentran:

- Asociación bidireccional de entidades (*Entity bean*) a las páginas.
- Eliminación de intermediarios (patrones de diseño)
- Reducción de descriptores y códigos gracias al uso de anotaciones.
- Soporte al manejo de conversaciones y de diferentes niveles de contexto
- Manejo de la *biyección*, que consiste en la asociación dinámica, contextual y bidireccional

entre las variables de los contextos y los atributos de los componentes de negocio.
- Portabilidad a cualquier servidor que soporte *JSF* y *EJB 3.0*.

5. CASO DE ESTUDIO

Se desarrolló el prototipo de una aplicación de software para hacer seguimiento y control de las comisiones de estudio y permisos académicos en una institución de educación superior [9]. En los siguientes apartados se describe el problema y la solución soportada en la arquitectura *JEE6*, utilizando *Richfaces 3.3* en la capa de presentación, *EJB 3.0* en la capa de negocio, *Hibernate 3.4* como mapeador objeto-relacional y *Seam 2.2* como framework integrador. Se utilizó *Oracle Express 10g* como base de datos relacional, *Jboss 4.2.3* como servidor de presentación y de componentes de negocio, y *Eclipse Indigo* como entorno integrado de desarrollo.

5.1 DESCRIPCIÓN DEL PROBLEMA

El acuerdo No. 009 de Diciembre 20 de 2007 [1] firmado por el Consejo Superior Universitario de la Universidad Distrital Francisco José de Caldas define las condiciones para que los docentes de planta adelanten programas de formación y actualización docente. Para acceder a un programa de magister, doctorado o postdoctorado el docente puede solicitar una comisión de estudio remunerada que tiene como contraprestación algunos compromisos de tiempo de servicio y producción intelectual, establecidos en el mismo acuerdo.

En la solicitud, trámite y aprobación de una comisión de estudio intervienen diferentes actores y unidades administrativas; una vez aprobada, el docente está obligado a entregar periódicamente, al responsable que corresponda según la reglamentación, el soporte de la documentación que exige el acuerdo. Hoy en día dichos soportes se entregan en formato físico y no existe un registro automatizado de la entrega lo cual dificulta la verificación y consolidación de información para efectos de seguimiento.

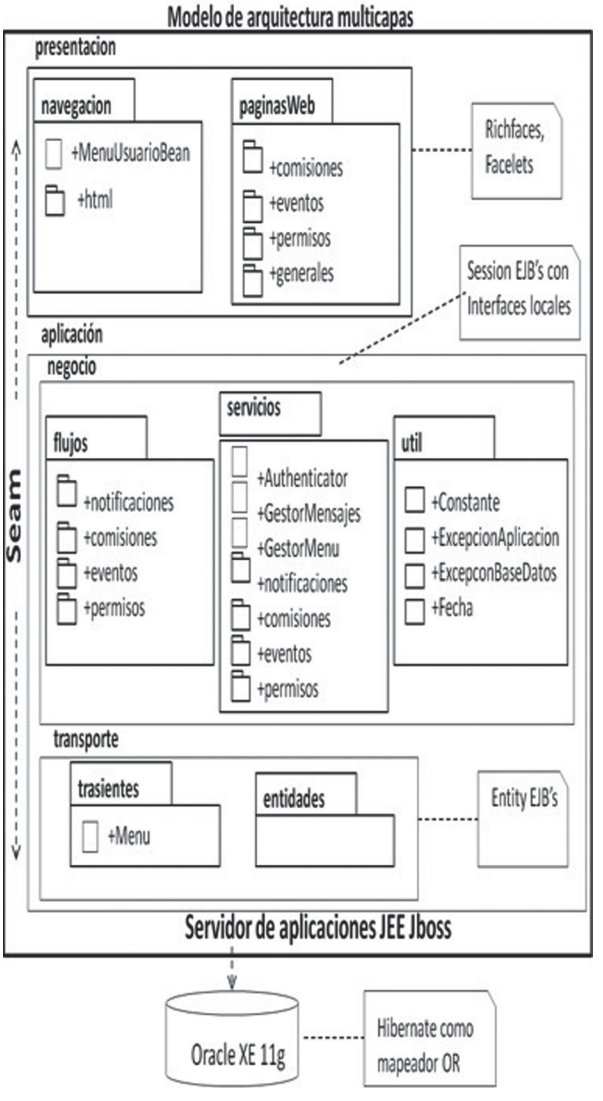
Con el fin de soportar el control automatizado de las comisiones de estudio, se diseñó e implementó un sistema de gestión bajo la arquitectura JEE6, que además de las comisiones incluye el trámite y asistencia a eventos académicos y el manejo de permisos docentes.

5.2 DISEÑO DE LA SOLUCIÓN

5.2.1 Modelo arquitectónico

Se diseñó un modelo de arquitectura multicapas por componentes como se ilustra en la figura 4.

FIGURA 4. Modelo de arquitectura del caso de estudio.



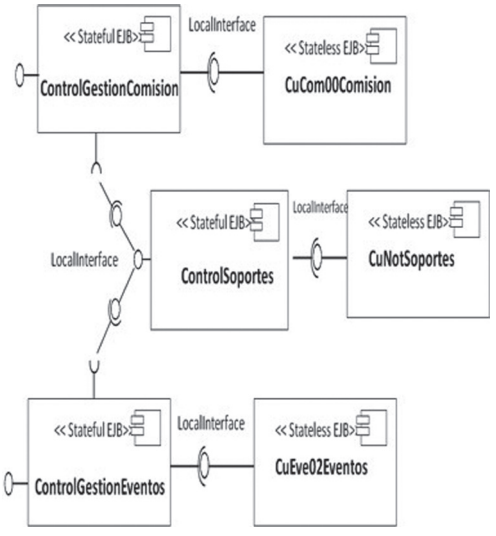
En la capa de presentación se incluyó un bean de manejo (*Managed Bean*) para construir dinámicamente el menú del usuario dependiendo de su rol (docente o funcionario). El menú está conformado por ítems cuyas acciones se configuran a partir de opciones declaradas en un archivo *XML*, y que en tiempo de ejecución invocan el servicio correspondiente ofrecido por un componente *EJB* de la capa de negocio. La capa de presentación también incluye las páginas *JSF* organizadas dentro de paquetes dependiendo del módulo funcional al cual pertenecen: comisiones de estudio, eventos académicos, permisos docentes y páginas generales cuya funcionalidad se puede incluir en cualquier módulo, por ejemplo, los datos personales de un docente.

La capa de aplicación está conformada por componentes *EJB*, agrupados funcionalmente dentro de paquetes: componentes de negocio, de transporte y utilitarios.

- Los componentes de negocio, *Session Beans*, a su vez se dividen en componentes controladores de flujo (de tipo *Stateful*) y componentes de servicio (de tipo *Stateless*) como se ilustra en la figura 5.

  Esta división se hizo con el fin de separar las responsabilidades del control de eventos asociados con la GUI (por ejemplo la construcción y manejo de listas de selección, el estado para habilitar o deshabilitar componentes gráficos, el manejo unificado de los mensajes de excepción y el control de su despliegue), de los servicios propios de la aplicación, como la verificación de las reglas de negocio. Dentro de esta capa también se encuentra el Authenticator, un componente Stateful y alcance de sesión, para validar el usuario y construir dinámicamente su menú de navegación.
- Dentro de los componentes de transporte, Entity Beans, están las entidades que representan los objetos persistentes de la aplicación que corresponden a tablas de la base de datos. Estos componentes fueron construidos y gestionados utilizando Hibernate. También se encuentra una entidad trasiente para construir el menú del usuario a partir de un archivo configurable XML.
- El paquete de utilitarios no está conformado por componentes EJB sino por clases Java que sirven para el tratamiento de las excepciones, manejo de fechas y construcción de algunos objetos de tipo código-descripción para el paso de información entre las capas.

FIGURA 5. Diagrama de componentes de la capa de negocio



5.2.2 Implementación de la solución JEE e integración con los frameworks de desarrollo

En la capa de presentación se utilizó *Richfaces 3.3*, y *Facelets* [5], un framework que permite definir plantillas con secciones predeterminadas como encabezado, contenido y pie de página, que son luego redefinidas por cada página. Se hizo uso de *Seam* para la construcción de decoradores de los componentes de la GUI, validadores y control de conversaciones. Se empleó el lenguaje *Expression Languaje* [5], para acceder a las propiedades y métodos de los *EJB* dentro de las páginas.

En la capa de aplicación, encargada de implementar el control y la lógica de negocio, se definieron componentes *EJB 3.0* de sesión, con interfaces locales. En la figura 5 se representa un subconjunto de dichos componentes y la interacción entre ellos mediante interfaces locales: los componentes responsables del control de flujo (ControlGestionComision, ControlGestionEventos, ControlSoportes) se definieron de tipo *Stateful* para mantener el estado de algunos objetos durante la misma sesión de usuario, por ejemplo, listas de opciones; y los componentes de servicios de negocio (CuCom00Comision, CuEve02Eventos, CuNotSoportes) que implementan los requerimientos funcionales de la aplicación, se definieron de tipo *Stateless* para mantener el estado de los objetos involucrados sólo durante la invocación de un mismo método. En el diagrama de componentes se nota cómo los componentes de Comisión y de Eventos hacen uso de los servicios ofrecidos por el componente de Soportes encargado de gestionar los documentos físicos que requieren los dos primeros.

En la implementación de los componentes de negocio utilizó la capacidad de **biyección** mediante las etiquetas *@In* y *@Out* de *Seam* para inyectar dentro del contexto de ejecución los objetos requeridos y dejarlos actualizados nuevamente para ser accedidos por otros componentes (ver figura 6). También se utilizaron las etiquetas de Seam *@Begin* y *@End* para demarcar el inicio y fin de una conversación larga entre usuario y sistema, por ejemplo, para la creación de una comisión de estudio que involucra varias páginas (ver figuras 9, 10 y 11), al final de la cual se deben eliminar los objetos utilizados en la conversación.

En la capa de transporte, encargada del paso de objetos entre la aplicación y la base de datos, además de componentes EJB de entidad con las correspondientes etiquetas para el mapeo objeto-relacional (@Table, @ Id, @Column, @OneToOne, @ManyToOne, entre otras), se definieron consultas nombradas mediante la etiqueta @NamedQuery (ver figura 7).

FIGURA 6. Biyección de objetos dentro de un EJB.

```
package udfjc.fing.sicap.negocio.servicios.comisiones;
import java.io.Serializable;
...
@Stateful
@Name("cuCom00GestionarComision")
@Scope(ScopeType.SESSION)
@AutoCreate
public class CuCom00GestionarComisionBean implements
CuCom00GestionarComision, Serializable
{
    @Logger
     private Log log;

    @PersistenceContext
     private EntityManager em;

    @In(required = false)
    @Out(required = false)
     private FoFormacion foFormacion;

    @In(required = false)
    @Out(required = false)
            private GrDocente grDocente;

    @In(required = false)
    @Out(required = false)
      private GrInstitucion grInstitucion;
...
```

Fuente: [9].

Las consultas nombradas pueden recibir parámetros y retornan un objeto o una colección de objetos; son de uso frecuente sobre una entidad, por ejemplo, para consultar todos los docentes, o buscar un docente por su identidad. Las facilidades de consulta fueron implementadas utilizando la implementación de Java Persistence Query Languaje JPQL , ofrecida por Hibernate.

FIGURA 7. NamedQuerie´s dentro de un Entity bean.

```
package udfjc.fing.sicap.transporte.entidades;

import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;
...
@Entity
@Table(name = "GR_DOCENTE")
@NamedQueries(value = {
   @NamedQuery(name = "GrDocente.buscarAllDocentes", query
= "SELECT d FROM GrDocente d" ),
   @NamedQuery(name = "GrDocente.buscarDocenteXId",  query
= "SELECT d FROM GrDocente d" +
  " WHERE d.KNumiden = :pKNumiden" )})

public class GrDocente implements java.io.Serializable {
    private String KNumiden;
...
```

Fuente: [9].

FIGURA 8. Archivo XML con las propiedades de conexión a la base de datos.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE datasources
    PUBLIC "-//JBoss//DTD JBOSS JCA Config 1.5//EN"
    "http://www.jboss.org/j2ee/dtd/jboss-ds_1_5.dtd">
<datasources>

   <local-tx-datasource>
      <jndi-name>udfjc.fing.sicapDatasource</jndi-name>
      <use-java-context>true</use-java-context>
      <connection-
url>jdbc:oracle:thin:@localhost:1521:xe</connection-url>
      <driver-class>oracle.jdbc.OracleDriver</driver-class>
      <user-name>sicap</user-name>
      <password>sicap</password>
   </local-tx-datasource>

</datasources>
```

Fuente: [9].

El manejo de la persistencia se le delegó al Entity Manager, objeto ofrecido por el framework EJB 3.0 para manejar las entidades de negocio persistentes. El Entity Manager es configurado con las opciones dadas en un archivo XML en el que se incluye la URL del servidor de datos, el nombre del driver, usuario y password de la base de datos (ver figura 8). El Entity Manager ofrece servicios para gestionar las transacciones y manejar entidades persistentes: persist(objeto), merge(objeto), find(objeto.class, id), remove(objeto) y refresh(objeto).

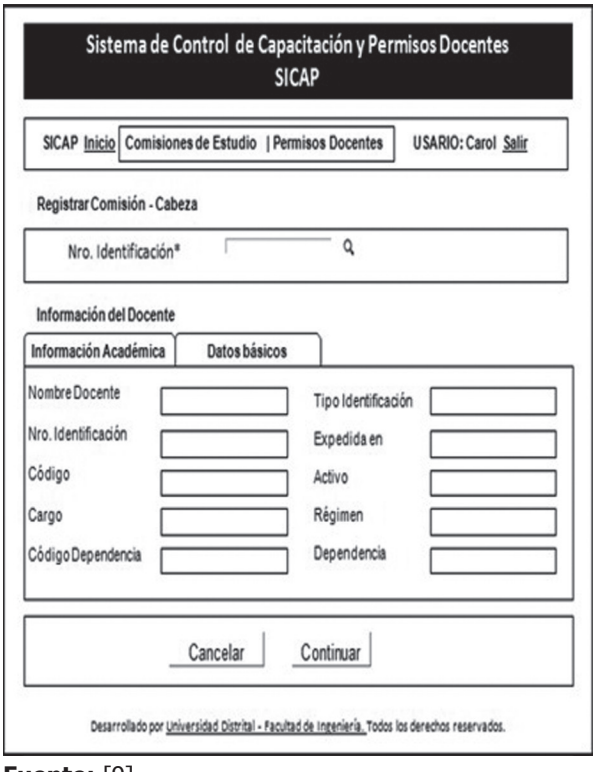
5.3 RESULTADOS OBTENIDOS

Se diseñó un modelo arquitectónico multicapas JEE para implementar el caso de estudio propuesto, con una clara separación y delimitación de responsabilidades por cada capa.

Se diseñaron los componentes de negocio utilizando Session Beans con interfaces locales para la exposición de los servicios, agrupándolos dentro de módulos funcionales (comisiones de estudio, eventos académicos, permisos docentes, notificaciones y gestión de soportes). Se definieron los objetos persistentes como Entity Beans correspondientes al modelo relacional de la base de datos y se mapearon convenientemente a la capa de negocio. Se construyó una interfaz de usuario amigable (ver figuras 9, 10 y 11) con un menú construido dinámicamente dependiendo del rol del usuario autenticado, incluyendo manejo de eventos y control de navegación aprovechando las capacidades de Richfaces.

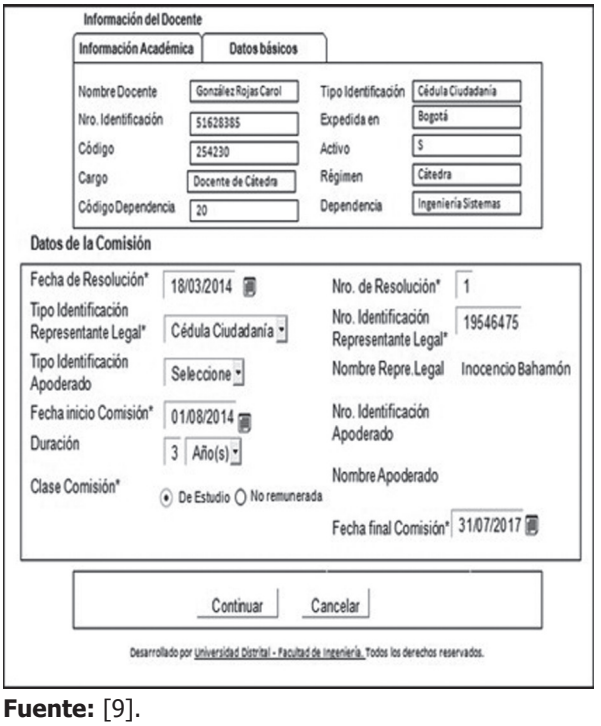
La integración entre las capas se hizo mediante las opciones y etiquetas de Seam, lo cual facilitó el manejo del ciclo de vida de los objetos durante la sesión y el transporte de dichos objetos entre las diferentes capas. Funcionalmente se hizo un manejo inicial de gestión documental para la administración de los documentos soporte asociados a los módulos de negocio y de gestión de notificaciones, alertas y correos electrónicos (para mayor detalle consultar [9]).

FIGURA 9. Página 1 de 3 para el registro de una comisión.



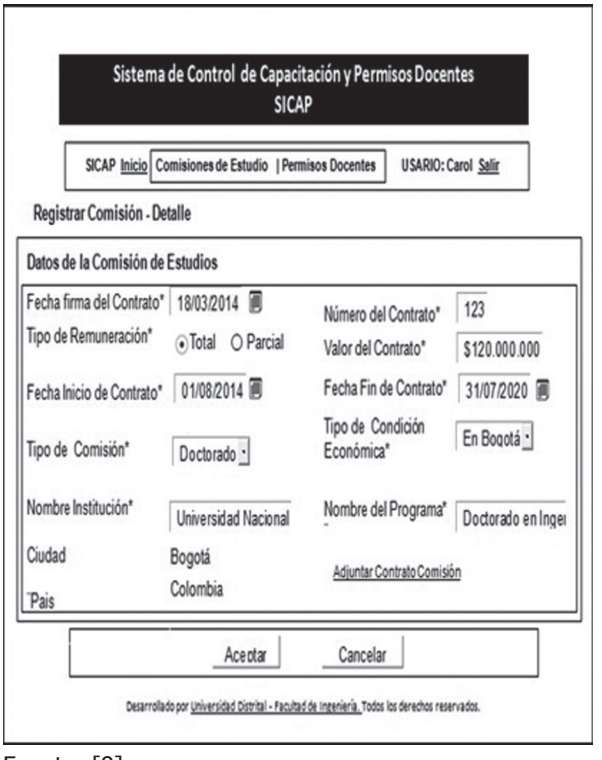
Fuente: [9].

FIGURA 10. Página 2 de 3 para el registro de una comisión.



Fuente: [9].

FIGURA 11. Página 3 de 3 para el registro de una comisión



Fuente: [9]

6. CONCLUSIONES

La dinámica en los procesos de negocio exige que las aplicaciones de software se adapten rápidamente a ellos con soluciones que den respuesta, no sólo a los requerimientos actuales, sino que también se anticipen a cambios futuros. Las integración con sistemas legados, la interconexión con sistemas externos de naturaleza diferente y con tecnologías heterogéneas, imponen retos que deben ser resueltos desde la arquitectura del sistema para garantizar que las soluciones propuestas sean desarrolladas, mantenidas y escalables dentro de las restricciones de tiempo y recursos que impone la industria. Es ahí donde los patrones arquitectónicos y el uso adecuado de frameworks de desarrollo aportan un gran valor.

En este artículo se ha descrito un ejemplo práctico de cómo los principios de la arquitectura de software y el uso de algunas herramientas de desarrollo pueden conjugarse para construir una solución efectiva que resuelva tanto requerimientos funcionales como no funcionales. La separación de responsabilidades en un modelo de arquitectura multicapas evidentemente promueve principios básicos del desarrollo como el bajo acoplamiento y la alta cohesión. Esto sumado al

diseño de componentes de software permite mayor rapidez en la construcción de aplicaciones mediante el ensamblaje de piezas de software reutilizables y facilita la escalabilidad y el mantenimiento.

Aunque los principios de la arquitectura JEE son precisos y muy claros los beneficios, su aprendizaje y puesta en práctica implica un gran esfuerzo inicial, que sin duda es compensado una vez se supere la curva de aprendizaje y se requiera modificar o incluir nueva funcionalidad. Los principios de la arquitectura, la separación de responsabilidades en capas y la integración de distintos frameworks compatibles con la especificación JEE, como algunos de los mencionados en este artículo, lleva a que se especialicen los roles en el proceso de desarrollo (arquitectos, expertos en el desarrollo de componentes de negocio, en diseño de bases de datos y en frameworks de presentación), y que se adopte como estrategia para la implementación exitosa de un sistema con requerimientos cambiantes que evolucionan en el tiempo.

7. REFERENCIAS BIBLIOGRÁFICAS

[1] Alcaldía Mayor de Bogotá D.C. (2007). Acuerdo No.009 de 2007. Por el cual se reglamenta el Estatuto Docente de la Universidad Distrital en cuanto a políticas y procedimientos para el apoyo a la Formación Postgradual de alto nivel a Profesores de Carrera y se dictan otras disposiciones. Recuperado (2011, mayo 25) de http://www.alcaldiabogota.gov.co/sisjur/normas/Norma1.jsp?i=37242

[2] Bijlsma, A., Heeren, B., Roubtsova, E., & Stuurman, S. (2011). Software Architecture. Free Technology Academy.

[3] Cheesman, J. & Daniels, J. (2001). UML Components. A Simple Process for Specifying Component-Based Software. EEUU: Addison-Wesley, 2001.

[4] Java Bean. (2014, 1 de agosto). En Wikipedia, la enciclopedia libre. Recuperado el 1 de agosto de 2014 a las 6:15 p.m. de http://es.wikipedia.org/wiki/JavaBean

[5] Jendrock, E., Cervera-Navarro, R., Evans, I. et al. (2013). The Java JEE 6 Tutorial. Redwood City: Oracle and/or its affiliates. Recuperado (2014,agosto 1) de http://docs.oracle.com/javaee/6/tutorial/doc/

[6] King, G., Bauer, C., Ebersole, S. et al. (2011) . Hibernate Developer Guide. Red Hat. Recuperado (2014, julio 30) de http://docs.jboss.org/hibernate/core/4.0/devguide/en-US/html_single/

[7] King, G., Muir, P., Richards, N. et al. (n.d.). Seam Contextual Components. A Framework for Entreprise Java. Recuperado (2014, julio 25) de http://docs.jboss.org/seam/2.3.0.Final/reference/en-US/html/

[8] Koziolek, H., Becker, S., Happe, J., & Reussne, R. (2008). Life-Cycle Aware Modelling of Software Components. Component-Based Software Engineering, 11th International Symposium, CBSE (págs. 278-284). Karlsruhe, Germany: Springer.

[9] Linares, J. (2012). Desarrollo del prototipo de software SICAP (Sistema de seguimiento de capacitación y permisos docentes) de la Universidad Distrital Francisco José de Caldas, bajo la metodología RUP y desarrollado cobre arquitectura web JEE. Informe de pasantía no publicado. Universidad Distrital, Bogotá, Colombia.

[10] Morrison, R., Balasubramaniam, D., Oquendo, F., Warboys, B., & Greenwood, M. (2007). An Active Architecture Approach to Dynamic Systems Co_evolution. First European Conference, ECSA (págs. 2-10). Madrid,Spain: Springer.

[11] Object Management Group (OMG). (2006). CORBA Component Model Specification; version 4.0; formal/06-04-01. Recuperado el 28 de 11 de 2014, de http://www.omg.org/spec/CCM/4.0/

[12] Rosen, M., Lublinsky, B., Smith, K. T., & Balcer, J. M. (2008). Applied SOA: Service-Oriented Architecture and Design Strategies. Indianápolis: Wiley Publishing.

[13] Silberschatz, A., Korth, H., & Sudarshan, S. (2011). Database System Concepts. Sixth Edition. New York:McGraw-Hill.

[14] Sistema heredado. (2014, 1 de agosto). En Wikipedia, la enciclopedia libre. Recuperado el 1 de agosto de 2014 a las 5:30 p.m. de http://es.wikipedia.org/wiki/Sistema_heredado

[15] Sommerville, I. (2011). Software Engineering - 9th edition. EEUU: Addison Wesley.

[16] Szyperski, C. (2002). Component Software - Beyond Object-Oriented Programming. EEUU: Addison-Wesley / ACM Press.

[17] www.iupuebla.com. (2003). Desarrollo de Software Basado en Componentes. Capítulo 1. Recuperado (2014, agosto1) de http://www.iupuebla.com/Ingenieria/Sis_compu/MA_sistemas/MA_PROGRAMACION_WINDOWS-sistema-basado-en-componentes-LAURA_MONARCA.pdf