UNIVERSIDAD DEL VALLE DE GUATEMALA
FACULTAD DE INGENIERÍA

DESARROLLO DE APLICACIONES EMPRESARIALES ORIENTADAS A OBJETOS CON J2EE Y HERRAMIENTAS DE CÓDIGO ABIERTO

TRABAJO DE GRADUACIÓN PRESENTADO POR
SERGIO GUSTAVO CHONG SEGURA
PARA OPTAR AL GRADO ACADÉMICO DE
MAESTRÍA EN TECNOLOGÍA Y CIENCIAS DE LA COMPUTACIÓN

GUATEMALA
2007

DESARROLLO DE APLICACIONES EMPRESARIALES ORIENTADAS A OBJETOS CON J2EE Y HERRAMIENTAS DE CÓDIGO ABIERTO

UNIVERSIDAD DEL VALLE DE GUATEMALA
FACULTAD DE INGENIERÍA

DESARROLLO DE APLICACIONES EMPRESARIALES ORIENTADAS A OBJETOS CON J2EE Y HERRAMIENTAS DE CÓDIGO ABIERTO

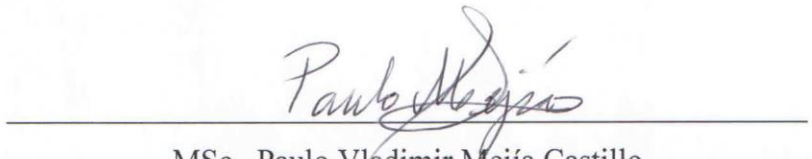
TRABAJO DE GRADUACIÓN PRESENTADO POR
SERGIO GUSTAVO CHONG SEGURA
PARA OPTAR AL GRADO ACADÉMICO DE
MAESTRÍA EN TECNOLOGÍA Y CIENCIAS DE LA COMPUTACIÓN

GUATEMALA
2007

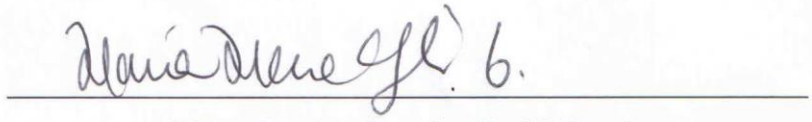
Vo. Bo.:

MSc. Paulo Vladimir Mejía Castillo

Tribunal examinador:



MSc. Paulo Vladimir Mejía Castillo



MBA. Maria Mercedes Zaghi García

MBA. Juan Antonio Cabrera Aguirre

Fecha de aprobación: Guatemala, 20 de noviembre de 2007

CONTENIDO

Capítulos		Página
	RESUMEN ……	vi
I.	INTRODUCCIÓN ……	1
II.	OBJETIVOS ……	2
III.	MARCO TEÓRICO …….	3
IV.	MODELO PROFESIONAL DE TRABAJO PROPUESTO …….	21
V.	PASOS GENERALES PARA LA APLICACIÓN DEL MODELO	35
VI.	APLICACIÓN DEL MODELO PROPUESTO ……	43
VII.	RESULTADOS Y DISCUSIÓN ……..	79
VIII.	CONCLUSIONES …….	83
IX.	RECOMENDACIONES ……	85
X.	BIBLIOGRAFÍA …….	87

RESUMEN

Las aplicaciones empresariales resuelven problemas de negocios gestionando la información que se maneja en ellas y poniéndola a disposición de sus usuarios a través de diferentes interfaces, permitiendo flexibilidad y capacidades de escalabilidad para soportar el crecimiento a largo plazo. La ingeniería de software orientada a objetos y los patrones de diseño de aplicaciones brindan el marco de trabajo para modelar y desarrollar sistemas basados en objetos independientes que se pueden implementar como componentes distribuidos de la edición empresarial del lenguaje Java sobre una plataforma basada en componentes de código abierto que gozan de buena reputación y que cuentan con servicios de consultoría y soporte técnico.

El modelo de trabajo propuesto se basa en un marco recursivo paralelo, en el que se modelan los requerimientos del negocio para dividir el sistema en componentes altamente independientes que se desarrollan simultáneamente para luego integrarlos en una aplicación final, que es implementada sobre un servidor de aplicaciones que provee todo el entorno para su ejecución y el acceso a la información contenida en el sistema de base de datos. El desarrollo de la aplicación es totalmente orientado a objetos y se obtienen dos conclusiones muy importantes. La primera es que los objetos del dominio del negocio son la base para todas las aplicaciones orientadas a objetos de una organización y en conjunto, conforman un subsistema en sí mismo. La segunda está relacionada con el hecho de que se aprecia una clara tendencia hacia el uso de aplicaciones desarrolladas en lenguajes orientados a objetos almacenando la información en sistemas de bases de datos relacionales de forma casi transparente para los desarrolladores, reduciendo considerablemente la cantidad de código escrito para las operaciones de almacenamiento y recuperación. Por último, se concluye que los recursos para impulsar el desarrollo de aplicaciones empresariales a través de emprendedores de tecnología, están disponibles junto con los marcos de trabajo para hacerlo profesionalmente.

I. INTRODUCCIÓN

Este trabajo trata sobre la propuesta de un modelo de trabajo profesional para el desarrollo de aplicaciones empresariales orientadas a objetos utilizando la plataforma empresarial del lenguaje Java sobre una arquitectura de implementación formada por componentes de software de código abierto.

El modelo se basa en la aplicación de un marco de trabajo recursivo paralelo obtenido de la ingeniería de software orientada a objetos, en el que se aplican técnicas de análisis, diseño y modelado empleando el lenguaje UML y un diseño de componentes y subsistemas siguiendo el enfoque modelo-vista-controlador, cuyo propósito es la creación de componentes altamente independientes implementados como subsistemas distribuidos, de tal forma que se obtengan las características esenciales de las aplicaciones empresariales, que son la flexibilidad, escalabilidad y disponibilidad para satisfacer los requerimientos de los procesos de negocios.

El trabajo se presenta en varios capítulos, iniciando con un marco teórico que contiene el desarrollo de los conceptos que sustentan al modelo, para luego pasar a un desarrollo de la propuesta, que es donde se especifican sus detalles, que incluyen al elemento principal de la propuesta, que es el patrón de diseño e implementación de los componentes de funcionalidad. Los detalles de implementación se ilustran por medio de un caso de aplicación, en el que se presentan los requerimientos de una serie de procesos de negocio, los cuales se modelan y se llega al diseño de un sistema dividido en componentes de funcionalidad altamente independientes, de los que se escoge uno para mostrar la forma en que se aplica el patrón de implementación y desarrollo propuesto en el modelo.

Al final se presentan las conclusiones obtenidas y las recomendaciones para la aplicación del modelo y extenderlo a un uso más amplio que genere mayores beneficios.

II. OBJETIVOS

A. GENERALES

1. Crear un modelo de trabajo profesional con un enfoque práctico y sencillo para desarrollar aplicaciones empresariales orientadas a objetos.

2. Mostrar el uso de componentes y herramientas de código libre como una alternativa viable para el desarrollo e implementación de aplicaciones empresariales.

B. ESPECÍFICOS

1. Mostrar cómo se puede hacer uso de un subconjunto de elementos del lenguaje UML para el diseño y modelado efectivo de una aplicación empresarial orientada a objetos.

2. Comprobar los beneficios que el proceso de desarrollo de software puede obtener de la orientación a objetos.

3. Establecer la arquitectura de una plataforma de implementación de aplicaciones empresariales en base a componentes de código libre.

4. Mostrar el uso de la plataforma J2EE para desarrollar una aplicación empresarial siguiendo el modelo vista controlador.

III. MARCO TEÓRICO

A. APLICACIONES EMPRESARIALES

1. DEFINICIÓN. Se da el nombre de aplicaciones empresariales al software desarrollado para resolver problemas de negocios y que contribuye a mejorar la productividad y eficiencia de una organización, las cuales se implementan en servidores de aplicaciones, donde son accedidas por múltiples usuarios simultáneamente a través de una red de computadoras.

2. CARACTERÍSTICAS.

a. *Soporte a procesos de negocio.* Automatizan o implementan procesos de negocio por medio de los cuales, una organización presta servicios a sus clientes internos o externos. Ejemplos de ello podrían ser el proceso a través del que una empresa realiza la venta de sus productos a sus clientes o el proceso necesario para obtener los informes financieros que permitan medir el desempeño de una organización.

b. *Manejo de información persistente.* Keith y Schincarlol (2006:1) destacan que la información manejada por este tipo de aplicaciones no es volátil y se almacena en sistemas de bases de datos que proveen seguridad, disponibilidad y eficiencia para su almacenamiento y recuperación bajo demanda, permitiendo que dicha información sea compartida por múltiples usuarios para diferentes propósitos.

c. *Acceso a través de múltiples interfaces.* Prestan servicio a diferentes tipos de usuario por medio de interfaces de acuerdo a las necesidades de cada uno en particular (Mukhar;Zelenak:2006:1), pudiendo ser usuarios internos trabajando en un computador de escritorio, usuarios externos accediendo a servicios por medio de un navegador de Internet o usuarios móviles accediendo a la aplicación por medio de un equipo portátil tipo PDA.

d. *Escalabilidad.* Poseen la capacidad de crecer en su infraestructura para incrementar sus niveles de servicio a más usuarios y mayor volumen de información, sin que su rendimiento sea degradado.

e. *Disponibilidad.* Cuentan con mecanismos que permiten que el servicio que prestan no sea interrumpido cuando algún componente de software o hardware presenta alguna falla.

B. PROGRAMACIÓN ORIENTADA A OBJETOS

1. ENFOQUE.

La orientación a objetos es un paradigma de programación que se enfoca en la resolución de un problema dividiéndolo en objetos, de forma que cada objeto funcione como un componente independiente. Un objeto es un elemento del programa que posee sus propios datos y su propio funcionamiento, es decir que está formado por datos que lo describen y funciones que implementan su comportamiento.

Según Tony Sintes (2002:11), definir un programa en términos de objetos es una forma profunda de visualizar el software. La orientación a objetos obliga a ver el programa como un todo que está compuesto por una colección de objetos independientes que realizan una tarea específica y que interactúan entre sí por medio del paso mensajes.

2. PROPIEDADES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS.

a. *Encapsulamiento.* Es la propiedad que permite dividir un programa en componentes independientes. El encapsulamiento mantiene dicha independencia ocultando los detalles internos de cada componente colocándolos detrás de una interfaz externa que lista los servicios proveídos por el objeto.

b. *Herencia.* Es el mecanismo que permite obtener una nueva clase a partir de la definición de una clase preexistente. Cuando una clase se hereda a partir de otra,

todos los métodos y atributos que aparecen en la interfaz de la clase preexistente, aparecerán automáticamente en la interfaz de la nueva clase.

c. *Polimorfismo.* En términos de programación, el polimorfismo permite que un único nombre de clase o método se utilice para representar diferente código fuente que es seleccionado en tiempo de ejecución por un mecanismo automático de acuerdo al cumplimiento de ciertas condiciones, consiguiendo que un mismo nombre represente muchos comportamientos diferentes.

3. BENEFICIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS. Según Tony Sintes (2002:19), la programación orientada a objetos brinda los siguientes beneficios:

a. *Programación natural.* Permite escribir los programas en lenguajes de alto nivel empleando términos del problema que se está resolviendo. Adicionalmente permite modelar el problema a nivel funcional sin necesidad de conocer detalles sobre cómo funcionan los componentes a emplear en la solución.

b. *Confiabilidad.* Los programas orientados a objetos son bastante confiables debido a su naturaleza modular que permite hacer cambios a una parte del programa sin afectar a otras partes. La confiabilidad se incrementa al desarrollar componentes independientes que se desarrollan y prueban por separado y que se incorporan a la solución hasta que han superado todas las pruebas de funcionamiento.

c. *Reutilización.* Los objetos se desarrollan y validan aisladamente como componentes independientes para resolver un problema específico, lo que permite lograr un débil acoplamiento y una alta cohesión que permiten incorporarlos en otros programas evitando la necesidad de volver a desarrollar código fuente para resolver el mismo problema.

d. *Fácil mantenimiento.* El código fuente orientado a objetos ofrece facilidades para su mantenimiento ya que una funcionalidad está completamente

implementada en un solo lugar, por lo que las correcciones o mejoras que se hagan sobre ésta serán transparentes para los otros objetos, que automáticamente se beneficiarán de la misma.

e. *Extensibilidad.* La programación orientada a objetos ofrece varias alternativas como la herencia, el polimorfismo, la sustitución y la delegación para extender el código fuente existente e incorporar nuevas funcionalidades de forma fácil y transparente.

f. *Menor tiempo de programación.* La programación orientada a objetos reduce el tiempo de programación al proveer software confiable, reutilizable y fácilmente extensible. Otra característica que ayuda a reducir los tiempos de programación es el hecho de que los objetos se pueden desarrollar aisladamente, lo que permite construir múltiples objetos en forma simultánea para luego incorporarlos en una aplicación final.

C. INGENIERÍA DE SOFTWARE ORIENTADA A OBJETOS

1. INGENIERÍA DE SOFTWARE.

Sobre la definición de Ingeniería de Software, (Pressmann, 2002:14) la define como:

> «El establecimiento y uso de principios robustos de ingeniería a fin de obtener económicamente software que sea fiable y que funcione eficientemente sobre máquinas reales».

En la actualidad existen dos enfoques que figuran como los más usados y mayormente difundidos: la ingeniería de software convencional y la ingeniería de software orientada a objetos.

La ingeniería de software convencional se basa en el análisis estructurado de sistemas, el cual se enfoca en el análisis de los pasos necesarios para resolver un problema, dividiéndolo en unidades más pequeñas de procesamiento para facilitar su solución.

La ingeniería de software orientada a objetos se enfoca en analizar los datos que intervienen en el problema y su comportamiento para crear objetos que luego se usarán como componentes de la solución del problema.

2. EL PROCESO DE LA INGENIERÍA DE SOFTWARE ORIENTADA A OBJETOS. Desde el surgimiento de la ingeniería de software, se han propuesto diversas metodologías y marcos de referencia que en su mayoría coinciden en la división del proceso en ocho fases conocidas como las fases clásicas del desarrollo de software, las cuales, según O'Docherty (2005:102), consisten en lo siguiente:

a. *Obtención de requerimientos.* Cumple con la función de descubrir qué es lo que se tiene que lograr con el nuevo sistema y se divide en dos aspectos: modelado del dominio y modelado de requerimientos. El modelado del dominio consiste en comprender el contexto en el que el software operará y se hace identificando el área del negocio a la que el sistema apoyará para limitar el análisis hacia dicha área y su relación con las demás. El modelado de requerimientos del sistema consiste en la especificación y documentación de las capacidades que el sistema tendrá.

b. *Análisis.* Significa comprender los detalles del problema con el que se está tratando para poder diseñar una solución y consiste en identificar con claridad cuáles son las entidades relevantes, sus propiedades y sus interrelaciones.

c. *Diseño.* En esta fase se trabaja en el diseño del sistema a desarrollar, dividiéndolo en los subsistemas de software que cumplirán una función específica.

d. *Especificación.* La especificación se usa para describir el comportamiento esperado de los componentes de software del sistema. Una especificación es la descripción de las condiciones que se tienen que cumplir y mantener antes y después de la ejecución de una función perteneciente a un componente de software.

e. *Desarrollo.* Es la fase en la que se escribe el código para construir cada uno de los componentes del software de acuerdo al diseño.

f. *Prueba.* Cuando el software está completo, debe ser probado para verificar que cumpla con los requerimientos y muestre el comportamiento deseado.

g. *Distribución.* Consiste en la entrega del hardware y software para ponerlo a disposición de sus usuarios finales y todas las actividades de capacitación y carga de información para que el sistema inicie su ciclo de vida productiva.

h. *Mantenimiento.* Consiste en todas las actividades necesarias para la incorporación de nuevas funcionalidades al software, así como también la corrección de los errores encontrados al estar en funcionamiento.

3. MODELOS PARA EL PROCESO DE DESARROLLO DE SOFTWARE. La ejecución secuencial de las ocho fases clásicas del desarrollo de software, ejecutando cada una de ellas solamente una vez se conoce como la metodología de la catarata y se basa en el supuesto de que el software se desarrollará en un solo ciclo del proceso, lo cual podría ser funcional en aplicaciones pequeñas, pero es difícil poner en práctica en aplicaciones grandes debido a que se necesitaría demasiado tiempo para tener el producto terminado y a que no permitiría regresar a una fase anterior del proceso para hacer algún cambio.

La metodología en espiral consiste en la repetición del ciclo de la catarata más de una vez, logrando corregir las deficiencias del enfoque anterior pero con la desventaja de que para regresar a una fase anterior es necesario repetir todo el proceso completo, lo cual es demasiado inflexible y difícil de manejar en la práctica.

La metodología iterativa es una mejora de la metodología espiral y permite regresar a una fase anterior sin necesidad de terminar el ciclo completo, logrando flexibilidad y proporcionando un enfoque bastante razonable para tratar con el problema

pero aún tiene una deficiencia: se enfoca en construir el software como un todo completo sin considerar su división en unidades de menor tamaño que se irán entregando en varios incrementos hasta llegar a la funcionalidad deseada.

La metodología incremental se basa en la metodología iterativa dividiendo el sistema en unidades entregables para desarrollar una primera versión del software implementando la funcionalidad básica y entregando el resto de funcionalidades en versiones subsiguientes.

4. EL PARADIGMA ORIENTADO A OBJETOS. De acuerdo a lo expresado por (Pressmann, 2002:344), el proceso orientado a objetos se inicia con un análisis del dominio, en el que se identifican las clases básicas que representan todos los objetos del dominio del negocio. El proceso continúa descomponiendo el sistema en unidades independientes que se analizan y desarrollan en forma paralela.

5. MODELO DE DESARROLLO RECURSIVO PARALELO. El modelo de desarrollo recursivo paralelo se basa en el paradigma orientado a objetos y según (Pressman, 2002:355), se aplica por medio de las siguientes actividades:

1. Descomposición del problema en partes altamente independientes
2. Aplicación del mismo proceso de descomposición a cada una de las partes independientes para, a su vez, descomponerlas en sus respectivos componentes (la parte recursiva).
3. Aplicación del proceso de forma simultánea sobre todos los componentes (la parte paralela).
4. Repetición del proceso hasta cumplir los criterios de finalización.

6. EL LENGUAJE DE MODELADO UML 2.0. El lenguaje de modelado unificado (UML por sus siglas en inglés) es un lenguaje gráfico que provee los mecanismos para visualizar, especificar, construir y documentar sistemas de software

(Eriksson, Hans-Erik, *et al*). Es administrado y apoyado por el *Object Management Group*, una institución de membresía abierta, no lucrativa que produce y mantiene especificaciones para el desarrollo de aplicaciones ínter operables para la industria de la computación.

UML surgió a finales de los noventa como resultado de la unificación de tres métodos para el análisis orientado a objetos (Pressmann:2002:363), y no es una metodología en sí misma, sino que consiste en un amplio conjunto de símbolos y diagramas cohesivos que se utilizan en las fases de requerimientos, análisis y diseño de aplicaciones orientadas a objetos.

Desde su creación se han publicado varias versiones del lenguaje en las que ha ido evolucionando con respecto a la versión original. La versión más reciente es la 2.0 y ofrece trece tipos de diagrama que pueden usarse para modelar casi cualquier sistema. La especificación del lenguaje no está apegada a ninguna metodología en particular y no dice donde se debería de usar cada tipo de diagrama. Tampoco obliga a usar todos los diagramas existentes, por lo que el diseñador es libre de escoger el más apropiado para cada fase del ciclo de desarrollo. Una práctica común consiste en seleccionar el subconjunto de los diagramas que serán suficientes para modelar el sistema.

D. EL LENGUAJE JAVA Y SU EDICIÓN EMPRESARIAL

1. EL LENGUAJE JAVA.

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems, cuyo uso se ha extendido ampliamente en los últimos años debido a que posee características que lo hacen ideal para desarrollar todo tipo de aplicaciones. Mike Parr y Douglas Bell en (Bell;Parr:545) describen sus características más importantes de la siguiente manera:

a. *Orientación a objetos.* Java fue creado completamente orientado a objetos. No es un lenguaje al que se le haya agregado esta orientación después de haber sido creado.

b. *Independencia con respecto a la plataforma operativa.* Los programas pueden ejecutarse en casi todas las plataformas operativas sin necesidad de modificarlos. Esto se logra por medio de un componente conocido como "maquina virtual de java" que existe para la mayoría de sistemas operativos, el cual interpreta el código de Java en forma idéntica en cada uno de ellos.

c. *Compatibilidad con Internet.* Cuenta con características que facilitan el desarrollo de programas para ejecutarse en navegadores de Internet.

d. *Extensibilidad por medio de bibliotecas.* Java es un lenguaje pequeño y la mayor parte de su funcionalidad se obtiene por medio de bibliotecas, las cuales se pueden adquirir comercialmente o como componentes de código abierto.

e. *Java Foundation Classes (JFC).* Consisten en un conjunto de librerías de clases desarrolladas para extender las capacidades del lenguaje Java y facilitar el trabajo de los programadores evitándoles la necesidad de desarrollar clases para el manejo de operaciones de bajo nivel. Estas librerías se combinan con otras como Swing y AWT para crear interfaces gráficas de usuario robustas y con múltiples funcionalidades que se conocen como interfaces de cliente grueso por la cantidad de código que involucran y por la cantidad de recursos que necesitan para su ejecución.

2. LA EDICIÓN EMPRESARIAL DE JAVA (J2EE). J2EE es una extensión de la plataforma Java que fue creada para desarrollar aplicaciones empresariales bajo una arquitectura distribuida basada en componentes de software ejecutándose sobre un servidor de aplicaciones. Java Edición Empresarial incluye las siguientes especificaciones que se detallan en (Mukhar, Kevin, *et. al*:2006:13) como se describe a continuación:

a. *Java Servlets.* Un Servlet es un componente de Java que se implementa en el servidor y que se invoca como resultado de una petición de ejecución hecha por el cliente hacia el servidor de aplicaciones. El Servlet responde ejecutando alguna tarea que puede involucrar cálculos y accesos a otros recursos del servidor para luego devolver la

información por medio de una página HTML. Los Servlets permiten desarrollar aplicaciones Web complejas ya que cuentan con todo el potencial del lenguaje Java.

b. *JavaServer Pages (JSP).* Consisten en páginas HTML con código Java inmerso dentro de ellas por medio de marcas estándar propias de JSP o XML. Tienen características y capacidades similares a los Servlets y se ejecutan de la misma forma, con la diferencia de que en una JSP, el código de Java está dentro de ella y es traducido por el contenedor del servidor de aplicaciones la primera vez que la página es invocada. Al igual que los Servlets, el resultado generado durante la ejecución es devuelto por medio de una página HTML.

c. *JavaServer Faces (JSF).* JSF es una tecnología un poco más reciente que se usa en conjunto con Servlets y JavaServer Pages y que provee una interfaz de programación basada en componentes para la creación de interfaces de usuario que se ejecutan del lado del servidor.

d. *Java Database Connectivity (JDBC).* Es una interfaz de programación de aplicaciones que permite la ejecución de operaciones sobre sistemas de bases de datos desde el lenguaje de programación Java utilizando el lenguaje estructurado de consultas (SQL). JDBC es más comúnmente usado con sistemas de bases de datos relacionales pero su uso puede extenderse hacia cualquier sistema de almacenamiento de datos, siempre y cuando se cuente con las librerías específicas para dicho sistema.

e. *Enterprise Java Beans (EJB).* Los EJB son componentes de Java que se invocan de forma remota y que se usan para implementar la lógica del negocio. El objetivo de los EJB es proporcionar al programador un modelo que le permita concentrar su atención en el desarrollo de la lógica del negocio sin preocuparse de los problemas de bajo nivel de una aplicación empresarial, como el manejo de transacciones concurrentemente, seguridad, persistencia, etc. Existen tres tipos de EJB:

- *EJB de entidad:* Encapsulan los objetos persistentes de la capa de almacenamiento y proveen mecanismos para facilitar su manejo en las aplicaciones
- *EJB de sesión:* Se utilizan para implementar el comportamiento de una aplicación por medio del uso de los servicios proporcionados por otros componentes disponibles en el servidor.
- *EJB conducidos por mensajes:* Son componentes con funcionamiento asíncrono que utilizan el sistema de mensajería de Java para suscribirse a un tema o cola. Se activan al recibir un mensaje dirigido hacia dicho tema o cola.

f. *Persistence API.* Es una interfaz de programación de aplicaciones que forma parte de la especificación EJB 3.0 y que fue creada para simplificar el desarrollo de los componentes que manejan la información persistente almacenada en servidores de bases de datos. Encapsulan la información almacenada en las tablas relacionales, en objetos que luego se convierten en entidades persistentes que se son manejadas por un componente de la API que provee todas las operaciones de bajo nivel necesarias para interactuar con el servidor de base de datos. Entre sus características más importantes se encuentran las siguientes:

- *Manejo de meta data por medio de anotaciones*: La interfaz de persistencia cuenta con identificadores especiales que se incluyen en la definición de las clases que encapsulan las entidades de datos para especificar propiedades relacionadas con su almacenamiento en el servidor de bases de datos, tales como el nombre de la tabla y sus columnas, las relaciones existentes con otras tablas, clave primaria, etc.

- *Mapeo objeto relacional*: Provee mecanismos que facilitan la descomposición de los objetos para guardarlos en tablas relacionales y encapsularlos nuevamente al ser recuperados de la base de datos.

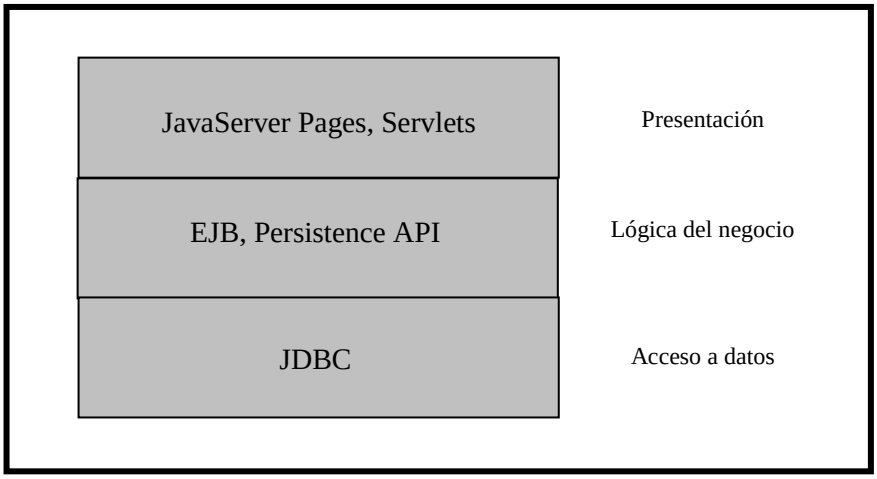
- *Lenguaje de consulta de datos*: Cuenta con un lenguaje propio para ejecutar consultas llamado EJB QL, que permite elaborar consultas complejas de forma sencilla.

E. ARQUITECTURA DE UNA APLICACIÓN EMPRESARIAL JAVA

La plataforma empresarial de Java permite el desarrollo de aplicaciones para ejecutarse en ambientes Web y por medio de aplicaciones con interfaces gráficas instaladas en la computadora del usuario. Los componentes de la aplicación se dividen en tres capas que implementan la interfaz del usuario, la lógica del negocio y el acceso a datos. En (Mukhar, Kevin, *et. al*:2006:24) se presentan algunos ejemplos de arquitecturas para esta plataforma, de las que se derivan las siguientes:

1. ARQUITECTURA DE SUBSISTEMAS PARA APLICACIONES WEB. Las aplicaciones Web son usadas para proveer servicios a usuarios no especializados del negocio y a menudo se les conoce como sistemas de autoservicio, en los que el usuario tiene poco conocimiento de los procesos internos del negocio.

Arquitectura de subsistemas para una aplicación Web

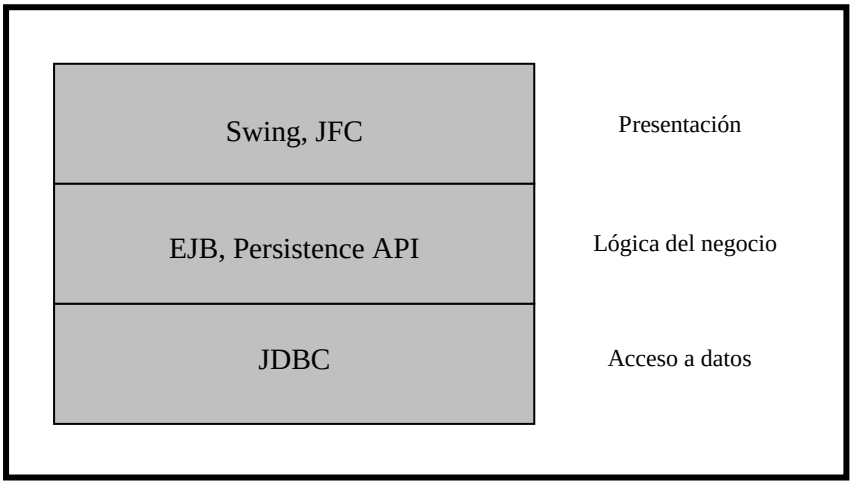


Los requerimientos de este tipo de usuario consisten en hacer uso de un servicio como el ingreso de un pedido de compra en línea, la consulta de un estado de cuenta y

otros similares que por lo general se hacen a través del Internet y no requieren de operaciones complejas ni de la instalación de una aplicación cliente en su computador para poder llevarlos a cabo.

2. ARQUITECTURA DE SUBSISTEMAS PARA APLICACIONES GUI. Las aplicaciones de interfaz gráfica de usuario están dirigidas a usuarios especializados del negocio que requieren de interfaces robustas que les provean de múltiples funcionalidades para realizar tareas complejas de los procesos de negocio y agilidad para el ingreso de altos volúmenes de información. Este tipo de aplicaciones requiere de la instalación de componentes de software en el computador y por lo general se acceden a través de redes de área local.

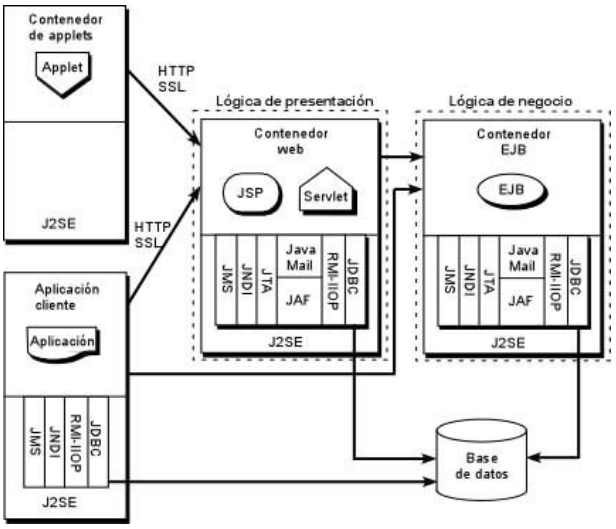
Arquitectura de subsistemas para una aplicación GUI



F. ARQUITECTURA DE LA PLATAFORMA DE IMPLEMENTACIÓN

Las aplicaciones J2EE se implementan sobre una plataforma cuyos componentes forman parte de una arquitectura distribuida, dividida en múltiples capas independientes que proveen servicios específicos que se pueden ejecutar en diferentes máquinas, lo que permite escalabilidad, alta disponibilidad, facilidad de mantenimiento y la posibilidad de incorporar componentes de distintos vendedores en una misma aplicación.

Arquitectura de componentes de la plataforma de implementación de una aplicación J2EE.



1. SISTEMA OPERATIVO. Provee del software necesario para el manejo y administración de los recursos de hardware, tales como la memoria central, procesador, dispositivos de almacenamiento físico y de entrada y salida. En una aplicación empresarial se requiere que permita escalabilidad por medio de la capacidad del hardware que sea capaz de administrar y de la posibilidad de trabajar en forma distribuida con otras instancias similares para formar clusters de procesamiento en paralelo.

2. SERVIDOR DE BASE DE DATOS. También conocido como Sistema Gestor de Bases de Datos (SGBD), es el encargado de proveer de toda la funcionalidad para el almacenamiento físico y la administración de la información usada por la aplicación.

3. SERVIDOR DE APLICACIONES. Los componentes de Java Empresarial se ejecutan en el servidor por medio de contenedores que consisten en un conjunto de aplicaciones que proveen todo el entorno para las tareas de bajo nivel y las interfaces necesarias para que un componente se pueda ejecutar de forma segura. Los contenedores están disponibles ejecutándose en un componente de software llamado servidor de aplicaciones, el cual consiste en una aplicación que tiene la función de ser un servidor de publicación Web que a la vez cumple con la especificación proveída por Sun

Microsystems para la implementación de los contenedores de cada uno de sus componentes.

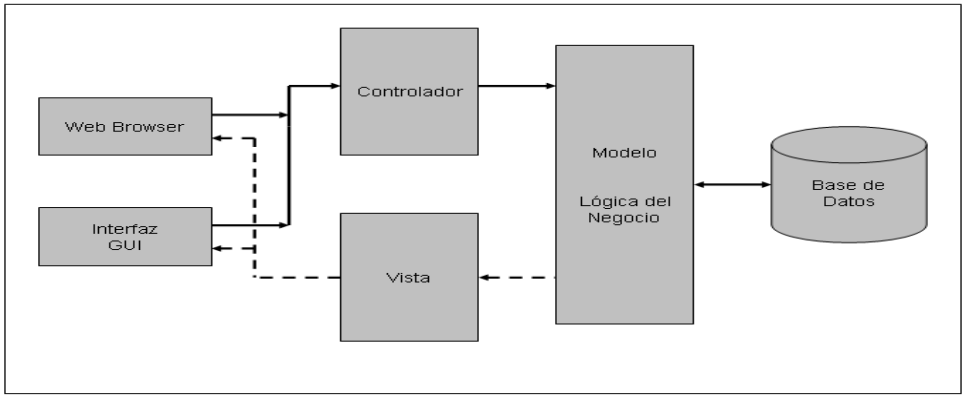
4. COMPONENTES DE LA APLICACIÓN. Los componentes de la aplicación son todas las clases desarrolladas para cada una de las capas de la aplicación, las cuales se implementan en la estructura de directorios del servidor de aplicaciones cuando se trata de componentes empresariales como JSP, Servlets y EJB, y localmente en la computadora cliente del usuario cuando son desarrollados utilizando Swing y JFC para aplicaciones GUI.

G. EL ENFOQUE MODELO VISTA CONTROLADOR (MVC)

El enfoque MVC (http://www.peachpit.com/articles/article.aspx?p=25464&rl=1) es un método de programación ampliamente usado en la programación de aplicaciones orientadas a objetos con Java para hacerlas fáciles de mantener, permitiendo su descomposición en capas para propósitos de escalabilidad.

El método divide los componentes del sistema en tres categorías: Modelo, Vista y Controlador. La siguiente figura ilustra un diagrama básico del método.

Enfoque Modelo-Vista-Controlador



1. MODELO. Cumple con la función de implementar el manejo de la información de los objetos del negocio y el manejo de todas las transacciones que se

realizan como consecuencia de los servicios prestados por el sistema a sus usuarios. Hace uso del subsistema de acceso a datos que provee todos los mecanismos para el acceso eficiente de la información contenida en servidores de bases de datos, de forma segura y manejando la concurrencia para que pueda ser compartida y actualizada manteniendo su integridad y consistencia.

2. VISTA. Se le conoce como capa de presentación, debido a que cumple con la función de proveer los medios para que el usuario interactúe con el sistema por medio del ingreso y consulta de información. Incluye la parte del software que crea y ejecuta la interfaz del usuario para controlar y validar sus acciones.

3. CONTROLADOR. Contiene las clases que implementan el comportamiento de la aplicación y la realización de los servicios que el sistema brinda a los usuarios. Es un mecanismo de interacción entre la interfaz del usuario y el modelo que implementa la lógica del negocio.

H. HERRAMIENTAS DE CÓDIGO ABIERTO

1. DEFINICIÓN. La organización Open Source Initiative[1] define el término Open Source, como el software acompañado por un contrato de licencia que garantiza que cualquier persona tiene el derecho de usar, modificar y redistribuir el código fuente libremente. El modelo de distribución requiere que se cumplan nueve condiciones que garantizan que no existan restricciones para su uso.

El término open source fue utilizado por primera vez en 1998 por usuarios de la comunidad de software libre, con la finalidad de hacer una clara distinción entre el software de uso gratis y el software que permite tener acceso al código fuente para poder modificarlo y distribuirlo sin ninguna restricción. La idea detrás del open source es que cuando los programadores pueden modificar y distribuir el código fuente de un programa, éste evoluciona, se desarrolla y mejora. Los usuarios lo adaptan a sus necesidades,

[1] Se puede encontrar la información completa en http://www.opensource.org/

corrigen sus errores a una velocidad impresionante, mayor a la aplicada en el desarrollo de software convencional o cerrado, dando como resultado la producción de un mejor software (http://es.wikipedia.org/wiki/C%C3%B3digo_abierto).

2. BENEFICIOS. En los últimos años se han llevado a cabo estudios para determinar los beneficios de las aplicaciones de código abierto, entre los que se puede citar el que fue realizado a nivel mundial por la Unión Europea en el año 2002.[2] Los resultados publicados y la opinión de personas con amplio conocimiento sobre el tema coinciden en señalar que los beneficios van más allá de los costos y en su mayoría se centran en la calidad del software, los cuales se citan a continuación (http://open-source.gbdirect.co.uk/migration/benefit.html).

a. *Confiabilidad.* Es más confiable debido a que se encuentran menos defectos en la programación ya que se cumple la ley de Linus que establece que "dados muchos ojos, todos los errores serán obvios".

b. *Estabilidad.* Los cambios al software se hacen para responder a cambios o mejoras en procesos de negocio y no para satisfacer estrategias mercadológicas, lo cual sucede con menor frecuencia y existen menos restricciones de tiempo y costo durante su desarrollo.

c. *Auditabilidad.* Al disponer del código fuente, es posible auditarlo desde diferentes puntos de vista para comprobar su seguridad, flexibilidad, adaptabilidad a cambios, cumplimiento con estándares, etc.

d. *Costo.* Los costos de licenciamiento no existen o son bastante bajos en comparación con los del software propietario, lo cual permite hacer una mejor

[2] Free/Libre and Open Source Software: Survey and Study. © 2002 International Institute of Infonomics. The Netherlands. http://www.infonomics.nl/FLOSS/report/

distribución de los recursos económicos de un proyecto y asignarlos a otros factores críticos de éxito para obtener mayores beneficios y un mejor retorno de la inversión.

e. *Flexibilidad*. Debido a que no se busca obtener ventajas competitivas ni limitar el uso de otros productos, el software de código abierto es generalmente desarrollado empleando estándares de la industria y herramientas que permiten su interacción con aplicaciones de otros vendedores.

f. *Soporte y responsabilidad*. Muchos vendedores de aplicaciones de código abierto obtienen la mayor parte de sus ingresos de los servicios de consultoría y soporte que ofrecen. Adicionalmente, por tratarse de aplicaciones distribuidas de forma libre, no existen limitaciones para que dichos servicios sean prestados en forma exclusiva por sus autores, sino que cualquier consultor puede brindar soporte sobre las aplicaciones que recomienda.

3. SITUACIÓN ACTUAL Y TENDENCIA FUTURA. El uso de aplicaciones de código abierto se ha extendido ampliamente y en la actualidad muchas de ellas se han ido convirtiendo en estándares de la industria dentro de su categoría. Entre dichas aplicaciones se pueden citar el sistema operativo Linux, el servidor de publicación Web Apache, el servidor de bases de datos MySQL, el servidor de aplicaciones JBoss y los entornos de desarrollo Netbeans y Eclipse entre otros.

La tendencia es hacia un incremento en su uso y a su incorporación en tecnologías propietarias, prueba de ello es que algunas de las empresas de desarrollo de software más grandes del mundo (http://www.oracle.com/lang/es/technologies/open-source/index.html), actualmente están invirtiendo fuertes sumas de dinero en el patrocinio proyectos para desarrollar componentes de código abierto para ser incorporados en sus aplicaciones o para ofrecer a sus clientes la opción de usarlos para interactuar con sus tecnologías propietarias.

IV. MODELO PROFESIONAL DE TRABAJO PROPUESTO

A. DESCRIPCIÓN

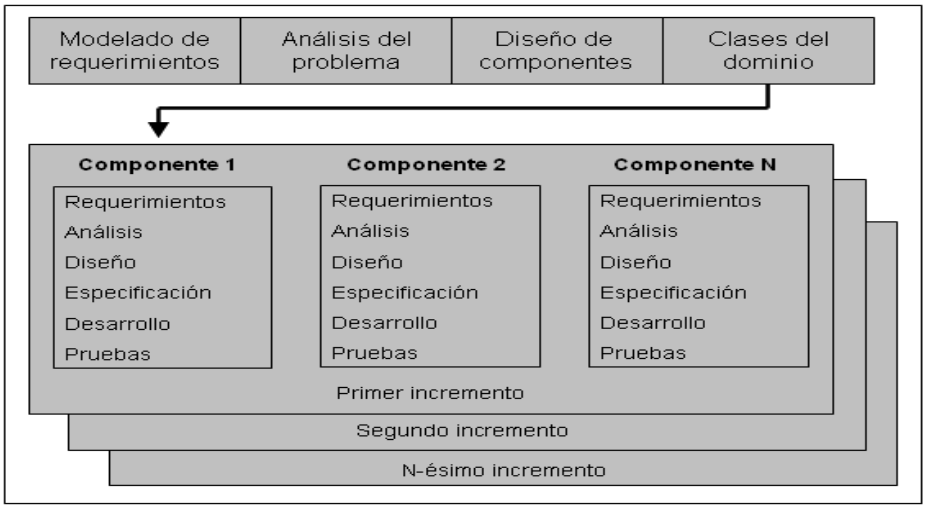
El modelo de trabajo profesional para el desarrollo de aplicaciones empresariales orientadas a objetos que se propone a continuación se basa en la aplicación de un marco de trabajo recursivo paralelo que se lleva a cabo en dos etapas principales:

1. *Análisis y diseño inicial.* Consiste en el modelado de los requerimientos del negocio y del sistema para hacer un análisis completo del problema y elaborar un diseño a nivel de sus componentes y subsistemas principales junto con las clases del dominio del negocio, incluyendo la planificación de su desarrollo siguiendo una estrategia incremental.

2. *Desarrollo de componentes.* Cada componente a incluir en el incremento en proceso se desarrolla como una unidad independiente, la que a su vez se puede dividir en subcomponentes a los que se aplica el mismo procedimiento. El desarrollo se lleva a cabo de acuerdo al paradigma orientado a objetos.

A continuación se presenta un esquema gráfico que muestra la secuencia de actividades de cada fase del modelo.

Fases del modelo de trabajo propuesto



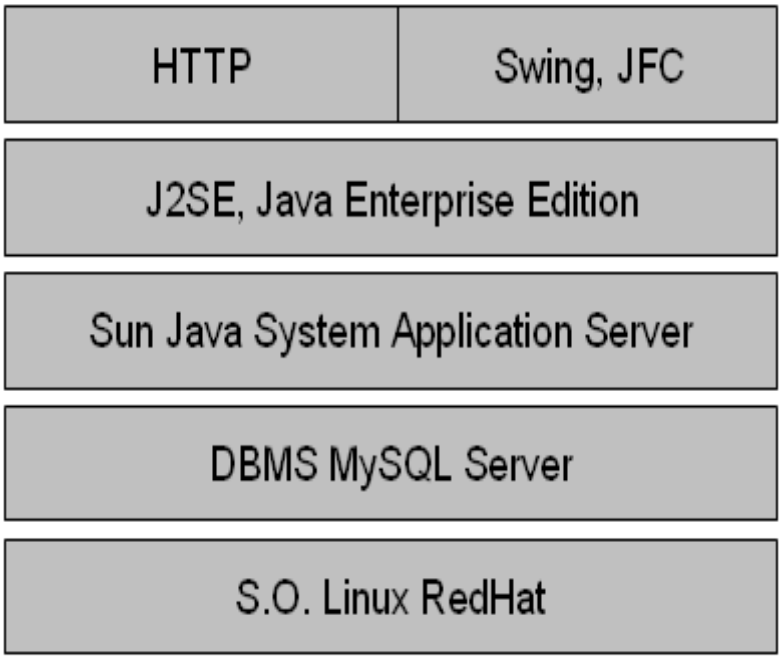
B. TABLA DE RESUMEN DEL MODELO PROPUESTO

Modelo propuesto				
Etapa	**Fase**		**Entregables**	**Detalles**
Análisis y diseño inicial	Requerimientos del sistema		Lista de casos de uso del sistema	Lista descriptiva
			Diagrama de casos de uso del sistema	Diagrama de casos de uso de UML
			Requerimientos de interfaces de usuario	Bosquejos de pantalla
	Análisis del problema	Análisis estático	Diagrama de clases	Diagrama de clases de UML
			Lista de atributos	Lista descriptiva
		Análisis dinámico	Realización de casos de uso	Diagramas de comunicación de UML
			Lista de operaciones y métodos de clases	Lista descriptiva
	Diseño	Componentes	Lista de componentes de funcionalidad altamente independientes	Lista descriptiva
		Subsistemas	Servlet controlador	Lista de mensajes
			Clase de servicios del negocio	Diagrama de clases de UML
			Componentes de la interfaz de usuario	Lista de componentes
			Realización de servicios del negocio	Diagramas de secuencia de UML
	Especificación		Especificación de clases de servicios	Predicados en lenguaje Java
			Especificación de clases del dominio del negocio	Predicados en lenguaje Java
Desarrollo de incrementos	Clases del dominio del negocio		Clases de entidades del dominio del negocio	Entidades API de persistencia
			Clases fachada para entidades del dominio del negocio	EJB de sesión
	Componentes		Clase de servicios del negocio	EJB de sesión
			Servlet controlador	Java Servlet
			Componentes de la interfaz de usuario	JavaServer Pages
	Prueba del incremento en proceso	Prueba	Inspección de código fuente	Listas de verificación
			Informe de resultados de pruebas funcionales	Informe descriptivo
		Corrección	Código fuente corregido	Clases de Java
	Distribución		Archivos empaquetados en el servidor de aplicaciones	Archivos ear, war, jar, xml

C. PLATAFORMA DE IMPLEMENTACIÓN CON COMPONENTES DE CÓDIGO ABIERTO

La plataforma de implementación que se propone en el modelo de trabajo está formada por componentes de código abierto cuya arquitectura se presenta a continuación:

Plataforma de implementación con componentes de código abierto



D. ANÁLISIS DE LOS COMPONENTES DE SOFTWARE DE LA ARQUITECTURA PROPUESTA

1. CRITERIOS DE EVALUACIÓN. Los componentes de la plataforma de implementación han sido seleccionados haciendo una comparación entre las opciones disponibles para cada uno de ellos considerando los siguientes aspectos:

- Características de utilidad para el modelo propuesto
- Disponibilidad de soporte y consultoría
- Caso de estudio

2. SISTEMA OPERATIVO LINUX RED HAT.

a. CARACTERÍSTICAS[3]

- Disponibilidad de la plataforma Java y del servidor de base de datos MySQL para ejecutarse en este sistema operativo
- Soporte para sistemas multiprocesador
- Tecnologías de código abierto probadas y maduradas a través del proyecto Fedora.
- Amplia disponibilidad de sistemas de hardware y periféricos certificados
- Disponibilidad de una edición empresarial del sistema operativo, diseñada para proveer escalabilidad y alta disponibilidad

b. CONSULTORÍA Y SOPORTE TÉCNICO

- Red Hat ofrece servicios y asistencia técnica 24 horas, 7 días a la semana con un tiempo de respuesta de una hora, proporcionados directamente por los desarrolladores del software y a través de socios independientes
- Cada versión mayor incluye interfaces de aplicación estables y siete años de asistencia técnica al producto

c. CASO DE IMPLEMENTACIÓN EXITOSA

CitiStreet - Linux Red Hat	
CitiStreet mejora su infraestructura de servicios a través de la Web y obtiene ahorros significativos	
Sector:	Financiero
Ubicación:	North Quince, Massachussets, USA

[3] http://www.redhat.es/rhel/features

Continuación

CitiStreet - Linux Red Hat	
Oportunidad:	CitiStreet estaba enfrentando una alta demanda de servicio en sus aplicaciones Web debido a un creciente incremento en su base de clientes y a la expansión de sus servicios por medio del Internet. Una migración de sus aplicaciones desarrolladas en C hacia Java Edición Empresarial había sido exitosa pero se necesitaba hacer mejoras significativas a la infraestructura sin sobrepasar el presupuesto disponible.
Software:	Servidor de aplicaciones JBoss 3.2, Linux Red Hat Enterprise 4, Java
Servicios:	Servicios Profesionales de Red Hat, Soporte de Red Hat a través de HP
	Consultoría y entrenamiento sobre JBoss, suscripción al servicio de soporte Platinum de Jboss
Beneficios:	Incremento significativo de los recursos de hardware sin incurrir en costos adicionales de licenciamiento
	Ahorros de cientos de miles de dólares al año en servicios de soporte y consultoría
	Infraestructura altamente disponible y escalable para soportar el crecimiento de la empresa a largo plazo
	Soporte de alta calidad disponible las veinticuatro horas del día, todos los días del año
Fuente:	http://www.redhat.com/solutions/successstories/financial/citistreet/

3. SERVIDOR DE BASES DE DATOS MYSQL SERVER.

a. CARACTERÍSTICAS [4]

- Disponibilidad para ejecutarse sobre el sistema operativo Linux Red Hat
- Disponibilidad de controladores para conexión a través de Java JDBC
- Procesamiento de transacciones y funcionalidades para implementar reglas del negocio por medio de disparadores y procedimientos almacenados
- Seguridad de acceso a la información
- Herramientas gráficas para la de administración de los servicios y bases de datos
- Disponibilidad de una edición empresarial que ofrece características de escalabilidad y alta disponibilidad

[4] http:/7www.mysql.com/products/enterprise

b. CONSULTORÍA Y SOPORTE TÉCNICO[5]

- Entrenamiento y certificación para administradores
- Consultoría sobre arquitectura y diseño, administración de bases de datos, ajuste para el rendimiento y optimización, alta disponibilidad, migración, etc.
- Servicios de soporte para resolución de problemas, consultas y acceso en línea a su base de conocimiento
- Alianza entre MySQL y Red Hat para ofrecer mejores servicios de soporte a las instalaciones que cuentan con ambos productos.

c. CASO DE IMPLEMENTACIÓN EXITOSA

Sony - MySQL Server	
Sony Internacional Europa reduce sus costos de base de datos con MySQL	
Sector:	Industria de electrónicos
Ubicación:	Stuttgart, Alemania
Oportunidad:	En un inicio, Sony había desarrollado una aplicación para el manejo del contenido de los resultados de las pruebas hechas a sus dispositivos electrónicos en proceso de desarrollo, empleando tecnología Microsoft con base de datos SQL Server y páginas ASP, para lo que contrató servicios de consultoría externa durante varios meses. A pesar de la fuerte inversión realizada, no se logró obtener los resultados deseados. Adicionalmente, los costos de licenciamiento fueron mucho más altos de lo que se había estimado originalmente. Sony comenzó a evaluar soluciones alternativas que fueran más efectivas en costo e inició la reescritura de su solución de manejo de contenido en un ambiente Linux con base de datos MySQL y lenguaje de programación PHP. El nuevo proyecto duró únicamente tres meses en desarrollarse consiguiendo resultados satisfactorios.
Software:	Linux, MySQL Server, Apache, PHP
Servicios:	Servicios de soporte de MySQL
Beneficios:	Reducción de tiempo de desarrollo
	Incremento en la eficiencia y rendimiento del sistema
	Reducción de costos de licenciamiento en un noventa por ciento
	Reducción de costos de administración y soporte en un cincuenta por ciento
Fuente:	http://www.mysql.com/why-mysql/case-studies/mysql-sony-casestudy.pdf

[5] http://www.mysql.com/services/

4. SERVIDOR DE APLICACIONES SUN JAVA SYSTEM APPLICATION SERVER.

a. CARACTERÍSTICAS [6]

- Desarrollado sobre estándares abiertos en lenguaje Java a través del proyecto GlassFish
- Disponibilidad para ejecutarse en el sistema operativo Linux Red Hat
- Cumplimiento de la especificación de contenedores para la edición empresarial de Java
- Procesamiento distribuido, tolerancia a fallos y balanceo de carga para la implementación de aplicaciones empresariales escalables y de alta disponibilidad.

b. CONSULTORÍA Y SOPORTE TÉCNICO

- Servicios de soporte Standard en horario hábil
- Servicios de soporte Premium veinticuatro horas todos los días del año

c. CASO DE IMPLEMENTACIÓN EXITOSA

Infosys Technologies Ltd. – Sun Java System Application Server	
Infosys Technologies Ltd., líder mundial en servicios de consultoría utiliza NetBeans 5.5 y Sun Java System Application Server para desarrollar un sistema empresarial de administración de activos	
Sector:	Tecnología
Ubicación:	Presencia a nivel mundial
Oportunidad:	Infosys comenzó a usar NetBeans 5.5 y Sun Java System Application Server para desarrollar una aplicación de demostración para entrenamiento interno pero al notar las facilidades proveídas por la herramienta, se tomó la decisión de usarla para desarrollar una aplicación empresarial para un ambiente de producción, que se desarrolló en un corto tiempo y con resultados que superaron las expectativas generando beneficios por encima de lo esperado

[6] Tomado de http://java.sun.com/javaee/downloads/index.jsp

Continuación

Infosys Technologies Ltd. – Sun Java System Application Server	
Software:	Sun Java System Application Server, NetBeans 5.5
Servicios:	Soporte en Sun Java System Application Server
Beneficios:	Menores tiempos de desarrollo de aplicaciones empresariales
	Simplificación del proceso de desarrollo de aplicaciones
	Fácil integración de componentes de sistemas
Fuente:	http://www.sun.com/customers/software/infosys.xml

5. ENTORNO DE DESARROLLO INTEGRADO NETBEANS.

a. CARACTERÍSTICAS

- Funcionalidad completa para desarrollar aplicaciones empresariales en lenguaje Java para diferentes servidores de aplicaciones incluyendo Sun Java System Application Server.
- Herramientas gráficas para el desarrollo de aplicaciones Web y GUI en forma visual
- Uso de patrones de diseño para Java
- Soporte para documentación con Javadoc
- Soporte para generación de código fuente de clases a partir de modelos UML
- Soporte para desarrollo de software basado en pruebas con JUnit
- Control de versiones de programas por medio del marco de trabajo de manejo de versiones concurrentes (CVS).
- Soporte colaborativo para trabajo en equipo
- Disponibilidad para diferentes sistemas operativos incluyendo Linux Red Hat
- Herramientas de administración de bases de datos por medio de JDBC

b. CONSULTORÍA Y SOPORTE TÉCNICO

- Publicación de actualizaciones para incorporación de mejoras y corrección de fallas
- Acceso a base de conocimiento en línea
- Manuales y tutoriales en línea
- Entrenamiento en línea

c. CASO DE IMPLEMENTACIÓN EXITOSA

1SYNC - NetBeans IDE	
1SYNC utiliza NetBeans para desarrollar aplicaciones y servicios Web para centros de procesamiento de datos	
Sector:	Tecnología
Ubicación:	USA
Oportunidad:	1SYNC necesitaba incorporar una herramienta que ofreciera un fuerte soporte al desarrollo de aplicaciones sobre el servidor de aplicaciones Java de Sun Microsystems.
Software:	NetBeans IDE 5.5, Java EE, Solaris, Sun Application Server
Servicios:	Entrenamiento para desarrolladores
Beneficios:	Mejora de los servicios a sus usuarios para recibir información de sus viajes y cambios de boletos
	Disminución de costos de licenciamiento
	Incremento en la productividad de los desarrolladores
Fuente:	http://www.sun.com/software/customers/sync.xml

E. ELEMENTOS A EMPLEAR EN EL CICLO DE DISEÑO Y DESARROLLO DEL SISTEMA

1. ACTORES. Un actor es cualquier entidad (persona, dispositivo o sistema de software) que juega un rol dentro del negocio.

2. CASOS DE USO. Los casos de uso son todas las actividades llevadas a cabo como parte de procesos del negocio y que son realizadas por los actores.

3. DIAGRAMAS DEL LENGUAJE UNIFICADO DE MODELADO UML. Se propone el uso de un subconjunto de diagramas del lenguaje UML, los cuales se usan en las fases de modelado de requerimientos, análisis y diseño del sistema. A continuación se presenta la lista de diagramas que se usarán, junto con una breve descripción del uso que se les dará.

Diagramas de UML	
Diagrama	**Uso**
Casos de uso	Representan gráficamente la interacción de cada actor con el sistema y la forma como se combinan los distintos casos de uso para satisfacer sus requerimientos
Actividades	Permiten representar los diferentes pasos de un proceso desde su inicio hasta su finalización, especificando la secuencia en que se realizan y los puntos de convergencia cuando se llevan a cabo simultáneamente
Clases	Representan las clases y sus relaciones. Permiten mostrar relaciones de herencia, agrupación, composición y especialización con cuantificadores de relación. Se complementan con las listas de atributos y mensajes que proporcionan la información necesaria para la definición de cada clase empleando elementos del lenguaje de programación
Comunicación	Permite modelar el comportamiento de los objetos de una clase y su comunicación con otros objetos por medio del paso de mensajes entre ellos. Su uso en el modelo propuesto consiste en el modelado del comportamiento del sistema para comprobar que los objetos a crear satisfacen los requerimientos del sistema.
Implementación	Permite modelar arquitectura de los componentes de hardware y software del sistema por medio de un diagrama que ilustra los componentes que existirán, los archivos de distribución que se tendrán que generar y los componentes de software que interactuarán para que el sistema funcione y esté disponible para los usuarios.
Secuencia	Permite modelar la realización de servicios del negocio por medio de la interacción secuencial entre objetos instanciados de las clases, en las que se muestra el flujo de la información en función del tiempo. Es la base para el desarrollo de las clases de servicios del sistema.

4. CLASES POJO. El término POJO (Plain Old Java Objects) se utiliza para describir las clases de Java que únicamente representan elementos del problema que se

está modelando y no contienen ninguna adición de métodos o propiedades provenientes de algún marco de trabajo.

5. JAVA BEANS. Son clases tipo POJO que modularizan y encapsulan la información contenida dentro de ellas para ser manejadas por los componentes empresariales de Java que requieren que se cumplan las siguientes características:

- Su método constructor no debe requerir parámetros
- No deben poseer atributos con acceso público, solamente privados o protegidos
- Para todos los atributos deben existir métodos set y get que sigan la nomenclatura estándar

6. CLASES DEL DOMINIO DEL NEGOCIO. Son las clases creadas para almacenar la información de los objetos del negocio, que en su mayoría se obtienen de los diagramas de casos de uso, donde por lo general se reconocen por tener un nombre propio. Estas clases deben diseñarse como JavaBeans para poder convertirse en entidades manejadas por la API de persistencia.

7. ENTIDADES. Las entidades son las clases que se producen al adicionar a las del domino del negocio, las anotaciones de la API de persistencia para especificar la información que las relacionará con los objetos de la base de datos relacional donde se almacenarán, que consiste en el nombre de la tabla, el identificador de llave primaria y las relaciones existentes con otras clases.

8. CLASES FACHADA. Son clases de soporte que se usan para proveer funcionalidades que faciliten el manejo de las entidades persistentes. NetBeans 5.5 posee plantillas para su generación a partir de la definición de las clases de entidad, que proporcionan los siguientes métodos:

- create(objeto): Crea un nuevo objeto dentro de la unidad de persistencia

- edit(objeto): Guarda los cambios realizados a un objeto
- destroy(objeto): Elimina un objeto de la unidad de persistencia, incluyendo todos los objetos con relación de agregación.
- find(Id): Devuelve un objeto cuyo contenido es el elemento que coincide con la llave primaria enviada como parámetro.
- findAll(): Devuelve una lista conteniendo todos los objetos de la entidad

9. PAQUETES DE CLASES. Los paquetes contienen grupos de clases relacionadas y se usan para hacer frente a la complejidad de un programa extenso. Consisten en estructuras de directorios donde se almacenan los archivos correspondientes a las clases que se desean agrupar. Permiten hacer una mejor organización del código fuente y tener un mejor control sobre las clases que tienen acceso a propiedades de otras clases relacionadas.

10. CLASES DE SERVICIOS DEL NEGOCIO. Son las clases desarrolladas para contener el código fuente que implementará los pasos e interacciones entre objetos para la realización de los casos de uso del sistema. Se implementan por medio de Enterprise Java Beans.

11. CLASES DE CONTROL. Son clases que no poseen estado y que cumplen con la función de separar el manejo de los servicios del negocio de los componentes de presentación para facilitar la división en capas de funcionalidad y permitir que la aplicación sea escalable y de fácil mantenimiento.

12. COMPONENTES DE LA INTERFAZ DE USUARIO. Consisten en clases o páginas JSP que se utilizan para presentar la información al usuario y proporcionarle los comandos y controles que necesitará para interactuar con el sistema.

13. ESPECIFICACIÓN DE INTERFACES DE CLASES. El modelo propuesto incluye la especificación de las interfaces de las clases por medio del diseño por contrato como se explica en (O'Docherty:2005:389), que consiste en la especificación de

contratos de prestación de servicios entre clases, definiendo a la clase que requiere el servicio como “el cliente” y a la clase que presta el servicio como “el proveedor”. El contrato especifica que ambos tienen obligaciones que cumplir con respecto a precondiciones, poscondiciones e invariantes, cuya no violación se debe garantizar de acuerdo a las obligaciones contractuales.

14. LISTAS DE VERIFICACIÓN DEL CÓDIGO FUENTE. Las listas de verificación del código fuente son plantillas que contienen la información derivada del diseño de todas las clases del sistema con sus atributos, métodos y la realización de los servicios del negocio. Las plantillas se usarán en la inspección del código fuente para comprobar que cumple con los detalles especificados en el diseño.

15. DOCUMENTACIÓN DE PROGRAMAS CON JAVADOC. Javadoc[7] es un estándar industrial para documentar clases de Java. La mayoría de los entornos de desarrollo integrados proveen funcionalidades especiales para su manejo y generan la documentación completa de un proyecto en formato HTML. Las unidades se documentan a nivel de clase y a nivel de métodos, existiendo etiquetas para especificar información relacionada con la unidad que se está documentando. A continuación se presenta una breve descripción de las etiquetas disponibles.

Etiquetas de Javadoc	
Etiqueta	**Descripción**
@author	Nombre del autor de la clase o método
@deprecated	Indica que la clase o función es antigua y no se recomienda su uso porque posiblemente desaparecerá en versiones posteriores
@param	Se utiliza para documentar la información sobre los parámetros de un método
@return	Se usa para documentar el valor de retorno de un método o función

[7] La información completa sobre Javadoc se encuentra en http://java.sun.com/j2se/javadoc/

Continuación

Etiquetas de Javadoc	
Etiqueta	**Descripción**
@see	Se usa para indicar referencias a otros métodos o clases que tienen relación con el elemento documentado
@throws	Indica las de excepciones lanzadas por el método
@version	Versión del método o clase

16. ARCHIVOS DE DISTRIBUCIÓN DE APLICACIONES JAVA. Las aplicaciones desarrolladas en lenguaje Java se distribuyen en archivos empaquetados cuyo contenido está compuesto por múltiples archivos de código compilado y de configuración, los cuales se deben implementar en la estructura de directorios del servidor de aplicaciones para que puedan ser accedidos por los usuarios. A continuación se describen los tipos de archivo existentes y el uso de cada uno de ellos.

Archivos de distribución de aplicaciones Java	
Formato	**Descripción**
JAR (Java Archive)	Se usa para empaquetar conjuntos de clases compiladas incluyendo descriptores de implementación
WAR (Web Application aRchive)	Se usa para empaquetar archivos de aplicaciones Web como descriptores en formato XML, clases compiladas, JavaServer Pages, Servlets, etc.
RAR (Resource Adapter aRchive)	Almacenan archivos XML, clases compiladas y otros objetos de aplicaciones J2EE
EAR (Enterprise Archive)	Incluyen archivos para aplicaciones empresariales Java, tales como descriptores en formato XML, clases compiladas, y archivos de distribución en formatos JAR, WAR y RAR

V. PASOS GENERALES PARA LA APLICACIÓN DEL MODELO

A. MODELADO DE REQUERIMIENTOS

1. MODELADO DE REQUERIMIENTOS DEL SISTEMA. El modelado de requerimientos comienza con un análisis de todos los procesos del área del negocio en que operará el sistema para dividirlos en casos de uso y determinar los que se incluirán como requerimientos del mismo. Los elementos a incluir en esta fase son los siguientes:

a. *Lista de actores del sistema.* De todos los actores que intervienen en los procesos del área del negocio, se deben identificar y extraer únicamente los actores que interactuarán directamente con el sistema propuesto.

b. *Lista de casos de uso del sistema.* Una vez obtenidos los actores del sistema, se deben incluir en la lista de casos de uso, únicamente los casos de uso que formarán parte del sistema propuesto. La lista puede incluir una narrativa de cómo ciertos casos de uso se integran para llevar a cabo un proceso del negocio, así como las relaciones existentes entre casos de uso. La descripción de cada caso de uso se debe expresar en términos de requerimientos sobre el funcionamiento del sistema.

c. *Diagrama de casos de uso del sistema.* Después de identificar los casos de uso que se incluirán en el sistema, se deben modelar por medio de un diagrama de casos de uso en UML, en el que se mostrará a los actores del sistema y sus asociaciones con casos de uso en particular.

d. *Detalles de casos de uso del sistema.* Para cada caso de uso del sistema se debe especificar la siguiente información:

   a. Número y título del caso de uso
   b. Relación con otros casos de uso

c. Secuencia de pasos del caso de uso

d. Requerimientos no funcionales

e. *Requerimientos para las interfaces de usuario.* Se debe proveer el diseño de las interfaces de usuario y los mecanismos sobre los que se implementarán. El requerimiento debe incluir las ilustraciones gráficas de cada una de ellas, una descripción sobre su funcionamiento y una sección de requerimientos adicionales.

f. *Planificación de incrementos.* Se debe asignar prioridades a cada uno de los casos de uso para planificar su inclusión en las diferentes entregas que se harán. La clasificación podría hacerse agrupando los casos de uso de acuerdo al número del incremento en que se incluirá su desarrollo.

B. ANÁLISIS DEL PROBLEMA

Para analizar el problema se toman los modelos de requerimientos del sistema obtenidos en el paso anterior para descomponerlos en los elementos esenciales sobre los que se basará la solución. Los entregables obtenidos en esta fase son el análisis estático y el análisis dinámico.

1. ANÁLISIS ESTÁTICO.

a. *Diagrama de clases y objetos.* Se utiliza el diagrama de clases proveído por UML para mostrar las clases del negocio a incluir en la solución junto con las interrelaciones entre ellas.

b. *Lista de atributos.* Al diagrama de clases se adjunta la lista descriptiva de los atributos de cada una de ellas.

2. ANÁLISIS DINÁMICO. El análisis dinámico consiste en la representación gráfica del paso de mensajes entre los elementos del diagrama de clases y cumple con la

función de mostrar la forma en que se darán las interacciones entre objetos para proveer la realización de los casos de uso.

a. *Diagramas de realización de casos de uso.* Se utilizan los diagramas de comunicación proveídos por UML para simular el envío de mensajes entre los objetos obtenidos al instanciar las clases incluidas en el análisis estático.

b. *Lista de operaciones y métodos de las clases.* Cada mensaje representado en el diagrama de realización de casos de uso corresponde a una operación de una clase, por lo que deberá agregarse a la clase correspondiente documentando la lista de parámetros que se necesitará para su funcionamiento.

C. DISEÑO

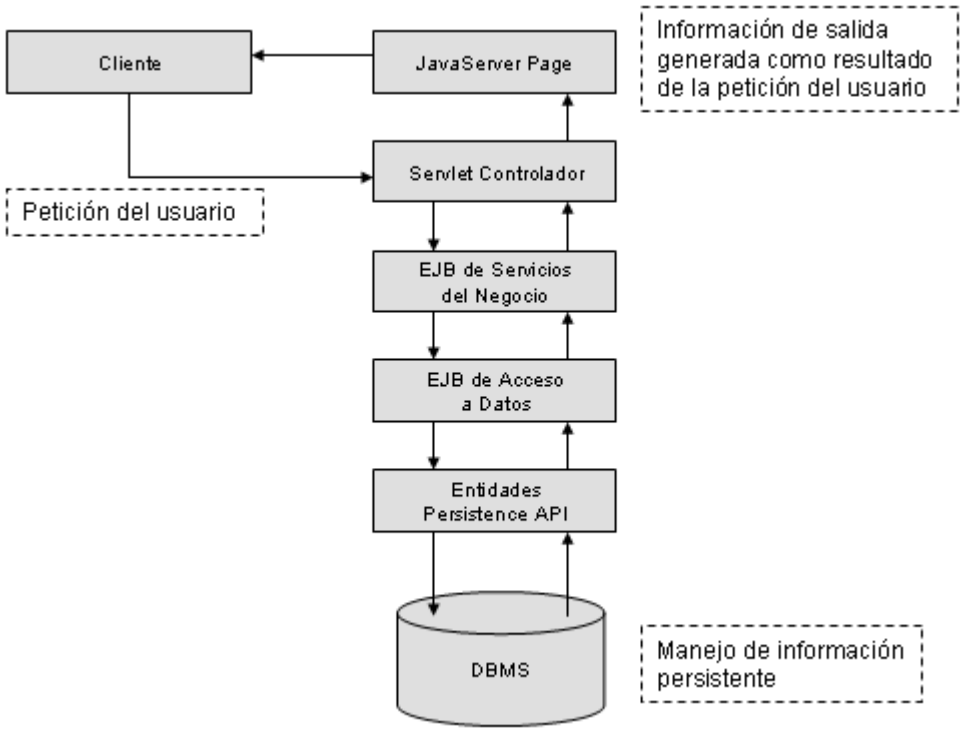
1. DISEÑO DE COMPONENTES DE FUNCIONALIDAD ALTAMENTE INDEPENDIENTES. El diseño de componentes de funcionalidad altamente independientes consiste en la división del sistema en componentes que agruparán casos de uso con relaciones de inclusión o extensión. Esta división permitirá desarrollar el sistema siguiendo el marco de trabajo recursivo paralelo.

2. DISEÑO DE SUBSISTEMAS DE COMPONENTES. Cada componente se diseña con base a una arquitectura estratificada basada en el modelo vista controlador, en la que se crean capas de funcionalidad cohesiva y débilmente acopladas en las que las capas inferiores prestan servicios a las capas superiores para implementar cada uno de los casos de uso del sistema. El diagrama muestra dicha arquitectura y su funcionamiento, que es el patrón general para el diseño de todos los componentes de funcionalidad. Los elementos a incluir en el diseño son los siguientes:

a. *Servlet controlador*. El servlet controlador cumple con la función de recibir las peticiones enviadas por el usuario desde la interfaz Web y enviar los mensajes correspondientes hacia la clase de servicios del negocio para la realización del caso de

uso requerido por el usuario. El lineamiento para su diseño se basa en la agrupación de casos de uso con relaciones de inclusión o extensión, con el propósito de crear un servlet controlador para cada componente de funcionalidad, que contenga los mensajes apropiados para su implementación.

Diagrama de funcionamiento de los subsistemas basado en el enfoque modelo vista controlador



b. *Clase de servicios del negocio*. La clase de servicios del negocio incluirá los métodos necesarios para la realización de cada caso de uso del sistema incluido en el componente de funcionalidad, por medio del paso de mensajes entre los objetos del dominio del negocio. Su funcionamiento se basa en el diagrama de realización de servicios del negocio.

c. *Componentes de implementación de la interfaz de usuario.* La interfaz de usuario se puede crear por medio de distintos componentes como servlets o páginas JSP para aplicaciones Web o por medio de clases de Swing para aplicaciones de consola, las

cuales se incluyen en el diseño por medio de una lista que describe el nombre, tipo, detalles de implementación y los casos de uso con los que están relacionados.

d. *Realización de servicios del negocio*. Para cada caso de uso se debe elaborar un diagrama de secuencia de UML que muestre la interacción secuencial entre los objetos que intervienen en su implementación, incluyendo a la clase de realización de servicios del negocio.

3. ESPECIFICACIÓN DE LAS INTERFACES DE CLASES. Se debe elaborar la especificación de las interfaces de cada clase del dominio negocio y de la capa de servicios representándolas por medio de predicados escritos utilizando la sintaxis del lenguaje Java siempre que sea posible. Cuando no sea posible la representación por medio de un predicado, se hará en forma descriptiva usando lenguaje común. La especificación de la interfaz de cada clase debe incluir los estados invariantes, las precondiciones y poscondiciones que se deben cumplir antes y después de hacer uso de un servicio prestado por la clase.

4. DESARROLLO. En la fase de desarrollo se escribe el código fuente de todas las clases incluidas en el diseño del sistema. La metodología a seguir se basa en el marco recursivo paralelo al permitir que los distintos componentes en que se divide el sistema se puedan desarrollar simultáneamente, permitiéndose también regresar a las fases de análisis y diseño en caso de ser necesario.

La fase de desarrollo de la aplicación se divide en dos etapas: planificación del incremento en proceso y desarrollo de programas.

a. *Planificación del incremento en proceso.* Consiste en identificar los casos de uso a incluir en el incremento en proceso y planificar su desarrollo de acuerdo a los siguientes pasos:

1. Desarrollo de las clases del dominio del negocio.

2. Desarrollo de los componentes de funcionalidad a incluir en el incremento en proceso, tomando en cuenta las prioridades asignadas a cada uno de ellos en la fase de requerimientos.

b. *Desarrollo de programas.* En esta fase se escribirá todo el código fuente de las clases a desarrollar de acuerdo a la planificación del incremento en proceso. Las prácticas que se deben seguir para el desarrollo son las siguientes:

- Instalación de un ambiente de desarrollo
- Creación de un proyecto de aplicación empresarial en NetBeans IDE seleccionando los tipos de aplicación que formarán el proyecto empresarial, la versión del kit de desarrollo de Java y el servidor de aplicaciones a emplear
- Creación de la unidad de persistencia que contendrá la información de conexión hacia el servidor de base de datos
- Desarrollo de las clases de entidades del dominio del negocio que contendrán las anotaciones con la información necesaria para hacer el mapeo objeto-relacional
- Creación de beans de sesión que contendrán las clases fachada para cada entidad
- Desarrollo de los componentes del sistema incluidos en el incremento en proceso.

MAPEO DE COMPONENTES DEL DISEÑO HACIA COMPONENTES DE IMPLEMENTACION			
Componente de diseño	**Componente de Implementación**	**Elemento del diseño**	**Paquete**
Clases del dominio del negocio	Entidad de la API de persistencia	Diagrama de clases del dominio del negocio, lista de mensajes	Business
Clases Fachada para entidades del dominio del negocio	Session Beans (EJB)	Diagrama de clases del dominio del negocio	Business

Continuación

MAPEO DE COMPONENTES DEL DISEÑO HACIA COMPONENTES DE IMPLEMENTACION			
Componente de diseño	**Componente de Implementación**	**Elemento del diseño**	**Paquete**
Clases de servicios	Session Beans (EJB)	Diagrama de clases de servicios, Lista de mensajes, Diagramas de realización de servicios del negocio	Server
Servlets controladores	Java Servlets	Lista de mensajes de servlets	Servlets
Interfaz de usuario	JavaServer Pages	Diseño de interfaz de usuario, Lista de requerimientos adicionales	No aplica

En la tabla anterior se presenta el mapeo de los componentes del diseño hacia los componentes de implementación del sistema. Se incluye una columna con el nombre del paquete recomendado para cada uno de ellos.

5. PRUEBA. La fase de prueba incluye la inspección del código fuente por medio de pruebas estructurales para comprobar que cumple con los estándares acordados y buenas prácticas de programación. Posterior a la inspección se hacen las pruebas funcionales que consisten en la elaboración de casos de prueba de los procesos del negocio para medir el desempeño del sistema.

a. *Inspección del código fuente*. Se realiza por medio de pruebas de caja negra y pruebas de caja blanca en las que se inspecciona el código fuente y se prueba el funcionamiento de cada unidad en forma independiente para encontrar errores y corregirlos antes de las pruebas funcionales.

b. *Pruebas funcionales*. Se hace por medio de la elaboración de casos de prueba en base a los requerimientos del sistema y cumplen con el objetivo de probar la robustez de la interfaz del usuario, el funcionamiento de los procesos del negocio dentro del sistema para verificar que el comportamiento que el usuario final experimentará y que las salidas generadas por el sistema sean correctas de acuerdo a los resultados esperados.

6. DISTRIBUCIÓN Y ENTREGA. La distribución y entrega del sistema se hace de acuerdo a los siguientes pasos:

a. Instalación y configuración del hardware donde operará el sistema
b. Instalación del sistema operativo Linux RedHat con la configuración adecuada para el hardware y el software a instalar.
c. Instalación y configuración del servidor de base de datos MySQL Server asignando los recursos de procesador, memoria y almacenamiento para satisfacer las necesidades de espacio, capacidad de usuarios y tiempos de respuesta requeridos por el sistema.
d. Instalación y configuración de la plataforma J2SE y la extensión J2EE
e. Instalación y configuración del servidor de aplicaciones Sun Java System Application Server
f. Generación de los archivos de distribución de las aplicaciones empresariales con su respectiva meta data e implementación de los mismos en la estructura de directorios del servidor de aplicaciones.
g. Configuración de los parámetros de conexión de las aplicaciones hacia los servidores de aplicaciones y de base de datos
h. Instalación del entorno de ejecución y clases de la interfaz de usuario en las computadoras cliente.

VI. APLICACIÓN DEL MODELO PROPUESTO

A. DESCRIPCIÓN DEL CASO DE ESTUDIO

El caso de estudio que se incluye en esta sección ilustra la forma como se aplica el modelo de trabajo propuesto al desarrollo de una aplicación empresarial, cuyos aspectos más relevantes son los siguientes:

- Análisis y diseño orientado a objetos
- División del sistema en componentes de funcionalidad que se desarrollan independientemente de los demás
- Implementación por medio de subsistemas débilmente acoplados y con funcionalidad cohesiva
- Aplicación del enfoque modelo-vista-controlador
- Desarrollo e implementación usando el lenguaje Java Edición Empresarial y componentes de código abierto

La solución se desarrolla aplicando un enfoque recursivo paralelo en el que se elaboran los elementos del modelado, diseño y desarrollo en las siguientes fases:

1. FASE I.
    - Modelado de requerimientos del sistema
    - Análisis estático y dinámico
    - División del sistema en componentes de funcionalidad altamente independientes
2. FASE II.
    - Diseño y desarrollo del subsistema de lógica del negocio
    - Diseño y desarrollo de los subsistemas del componente de funcionalidad llamado Test, debido a que es el más ilustrativo de la aplicación del modelo de trabajo

B. MODELADO DE REQUERIMIENTOS

Caso de aplicación: Certinet

Certinet es una institución que presta sus servicios de administración de exámenes de basados en computadora a distintas empresas afiliadas que cuentan con programas de certificación sobre los cursos que se imparten en sus centros de capacitación.

Un programa de certificación consiste en un examen que el aspirante toma en forma presencial, que se aprueba superando la marca mínima requerida. Los exámenes se realizan en la modalidad conocida como selección múltiple, en la que el sustentante recibe una lista de preguntas con cuatro posibles respuestas, de las que debe seleccionar la que considere correcta.

Para poder hacer uso de los servicios de Certinet, el afiliado presenta una solicitud de servicios donde suministra la información de la empresa, los programas de certificación que ofrece y los requisitos que un aspirante debe reunir para ser aceptado en el programa. El departamento de certificaciones verifica la información y suscribe un contrato de prestación de servicios con el afiliado, en el que se fija la tarifa que se cobrará por cada examen realizado de cada certificación. Al firmarse el contrato, el afiliado es aceptado en la empresa y presenta la base de datos de preguntas vigentes para todos los programas incluidos en el contrato. Una vez cumplidos todos los pasos anteriores, los exámenes están disponibles para ser administrados por Certinet.

El aspirante a un programa de certificación presenta una aplicación adjuntando las constancias que comprueban que reúne los requisitos establecidos para ser aceptado. La solicitud es evaluada y verificada por el departamento de certificaciones, que confirma o rechaza la inscripción.

Para tomar un examen, el aspirante se presenta al centro de evaluación, muestra un documento de identificación y realiza el pago correspondiente al valor del derecho a examen, en el que se le asigna un numero de cupón, el cual es ingresado en su cuenta para habilitarlo en el sistema y posteriormente pasa a la sala de evaluación donde se realiza el examen.

1. MODELADO DE REQUERIMIENTOS DEL SISTEMA

a. LISTA DE ACTORES DEL SISTEMA

Actor	Descripción
Usuario público	Usuario que tiene acceso al sistema por medio de un navegador de Internet que le permite consultar información sobre los programas de certificación disponibles y registrarse para aplicar a uno de ellos.
Usuario registrado	Usuario que ha proveído su información personal en el sistema para crear una cuenta de usuario. Posee una clave de acceso que le permite iniciar sesión en el sistema para aplicar a una certificación y tomar el examen correspondiente.
Departamento de certificaciones	Departamento de Certinet que se encarga de la creación de las cuentas de afiliados, sus programas de certificación y las inscripciones e ingreso de cupones de examen de los usuarios registrados.

b. CASOS DE USO DEL SISTEMA

Caso de uso	Detalles	
U1: Iniciar sesión	El usuario inicia sesión en el sistema ingresando su cuenta de usuario	
	Extendido por U3, U11, U16	
	1	El usuario ingresa su cuenta de correo electrónico
	2	El usuario ingresa su contraseña
	3	El usuario presiona el botón de iniciar sesión
U2: Registrar usuario	El usuario crea su cuenta ingresando su información en el sistema	
	1	El usuario selecciona la opción de registrarse en el sistema
	2	El usuario ingresa su información personal
	3	El usuario confirma el envío de su información
U3: Visualizar categorías	El usuario visualiza las categorías de certificación disponibles	
	Extendido por U4	
	1	El usuario selecciona la página de inicio
	2	El sistema le presenta la lista de categorías disponibles
	3	Extender con “visualizar certificaciones”
U4: Visualizar certificaciones	El usuario visualiza las certificaciones de una categoría	
	Extiende a U3. Extendido por U5	
	1	El usuario selecciona una categoría de certificación
	2	El sistema le muestra la lista de certificaciones de la categoría seleccionada
	3	Extender con "Ver detalles de certificación"
U5: Ver detalles de certificación	El usuario visualiza los detalles de una certificación	
	Extiende a U4. Extendido por U6	
	1	El usuario selecciona una certificación
	2	El sistema le muestra los detalles de la certificación y los requisitos para obtenerla
	3	Extender con "Aplicar a certificación"

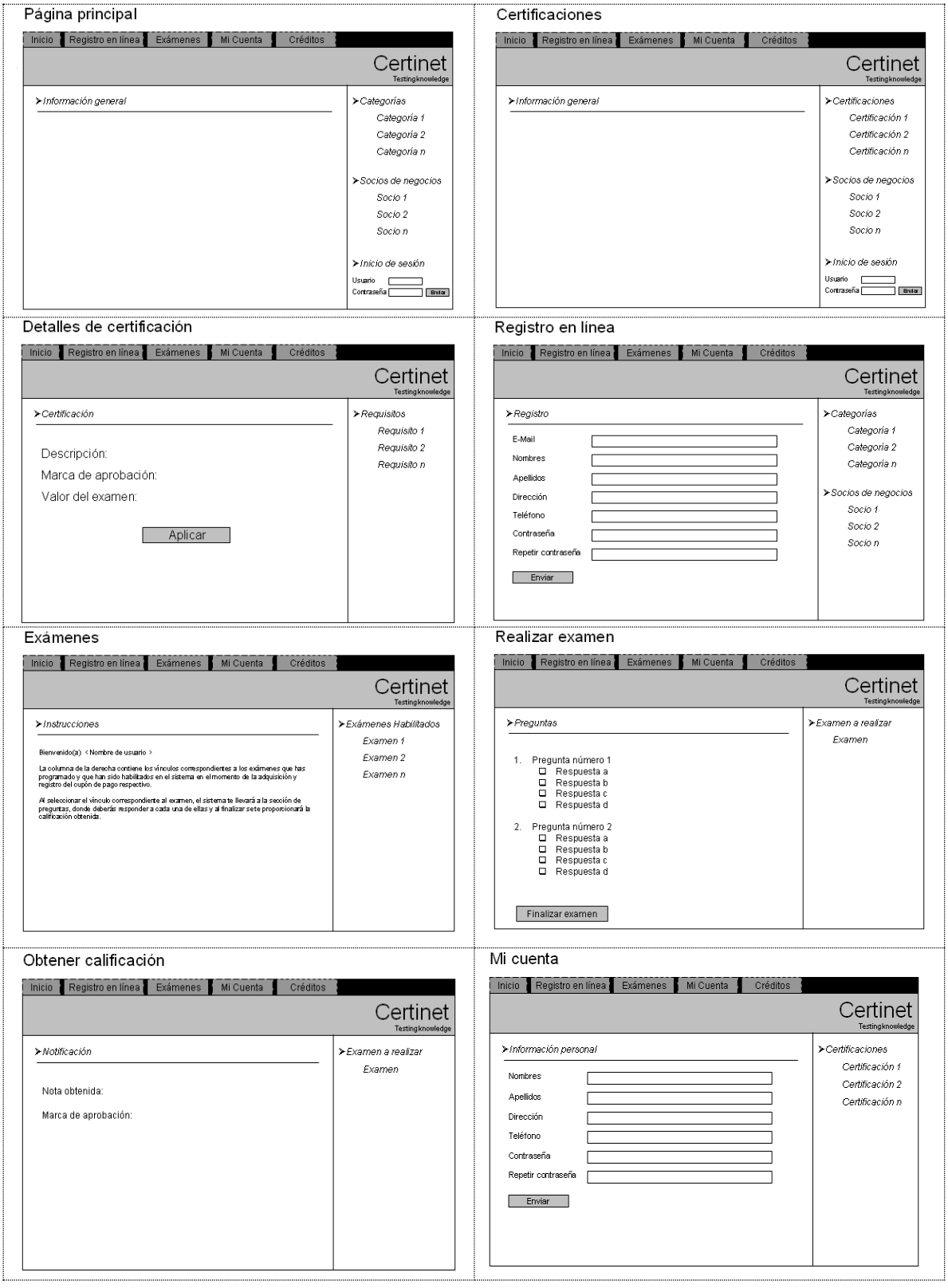
Continuación

Caso de uso	Detalles	
U6: Aplicar a certificación	El usuario aplica a una certificación	
	Extiende a U5	
	1	El usuario registrado selecciona la opción de aplicar a certificación
	2	El sistema le notifica si la aplicación fue exitosa
U7: Ver lista de exámenes habilitados	El usuario visualiza la lista de exámenes habilitados para ser realizados	
	Extiende a U1. Extendido por U8	
	1	El usuario registrado selecciona la opción de exámenes
	2	El sistema le muestra la lista de exámenes habilitados para ser realizados
	3	Extender con "Tomar examen"
U8: Realizar examen	El usuario visualiza la lista de preguntas del examen para realizar el examen	
	Extiende a U7. Incluye a U9	
	1	El usuario registrado selecciona un examen habilitado
	2	El sistema obtiene la lista de preguntas con sus posibles respuestas y la muestra al usuario
	3	Incluye a "obtener calificación"
U9: Obtener calificación	El usuario finaliza el examen y obtiene la calificación a la que se ha hecho acreedor	
	Incluido por U8	
	1	El usuario registrado finaliza el examen
	2	El sistema calcula el resultado y lo despliega al usuario
U10: Mi cuenta	El usuario registrado visualiza la información de su cuenta	
	Extiende a U1	
	1	El usuario registrado selecciona la opción de visualizar información de su cuenta
	2	El sistema el muestra su información personal y la de certificaciones a las que ha aplicado
	3	El usuario modifica su información personal y la envía al sistema
	4	El sistema el notifica si la información fue recibida satisfactoriamente

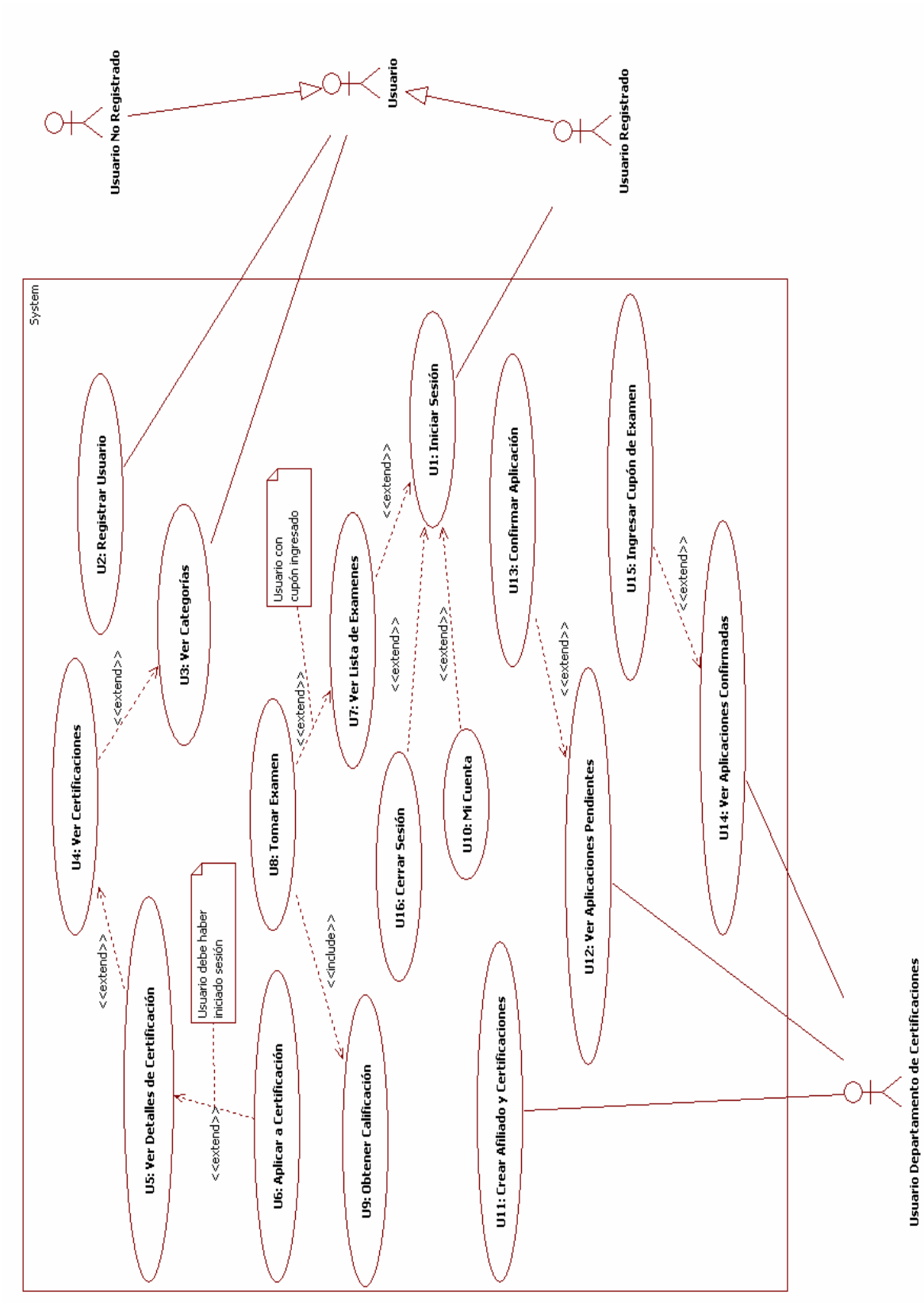
Continuación

Caso de uso	Detalles	
U11: Crear afiliado y certificaciones	El departamento de certificaciones crea cuentas de afiliado con sus certificaciones y la base de datos de preguntas y respuestas de cada una de ellas	
	1	El departamento de certificaciones selecciona la opción de crear afiliados
	2	El departamento de certificaciones ingresa la información del afiliado
	3	El departamento de certificaciones ingresa las certificaciones disponibles para el afiliado
	4	El departamento de certificaciones ingresa la base de datos de preguntas de cada certificación
	5	El usuario selecciona la opción de guardar la información
	6	El sistema le notifica si la información fue recibida satisfactoriamente
U12: Ver aplicaciones pendientes	El departamento de certificaciones visualiza la lista de aplicaciones hechas por los usuarios que están pendientes de confirmar	
	Extendido por U13	
	1	El departamento de certificaciones selecciona la opción de visualizar aplicaciones pendientes de confirmar
	2	El sistema le muestra la lista de aplicaciones pendientes de confirmar
	3	Extender con "Confirmar aplicación pendiente"
U13: Confirmar aplicación	El departamento de certificaciones visualiza los detalles de una aplicación e indica si la confirma o rechaza	
	Extiende a U12	
	1	El departamento de certificaciones selecciona una aplicación pendiente de confirmar
	2	El sistema le muestra los detalles de la aplicación
	3	El departamento de certificaciones ingresa el nuevo estado: confirmada o rechazada
	4	El departamento de certificaciones envía la información al sistema
	5	El sistema le notifica si la información fue recibida satisfactoriamente
U14: Ver aplicaciones confirmadas	El departamento de certificaciones visualiza la lista de aplicaciones confirmadas	
	Extendido por U15	
	1	El departamento de certificaciones selecciona la opción de visualizar aplicaciones confirmadas
	2	El sistema le muestra la lista de aplicaciones confirmadas
	3	Extender con "ingresar cupón de examen"
U15: Ingresar cupón de examen	El departamento de certificaciones ingresa un cupón de examen para una aplicación confirmada	
	Extiende a U14	
	1	El departamento de certificaciones selecciona una aplicación confirmada
	2	El sistema muestra los detalles de la aplicación y una casilla para ingresar el número de cupón
	3	El departamento de certificaciones ingresa el número de cupón y envía la información
	4	El sistema notifica al usuario si la información se recibió correctamente

c. REQUERIMIENTOS DE LAS INTERFACES DE USUARIO



d. DIAGRAMA DE CASOS DE USO DEL SISTEMA



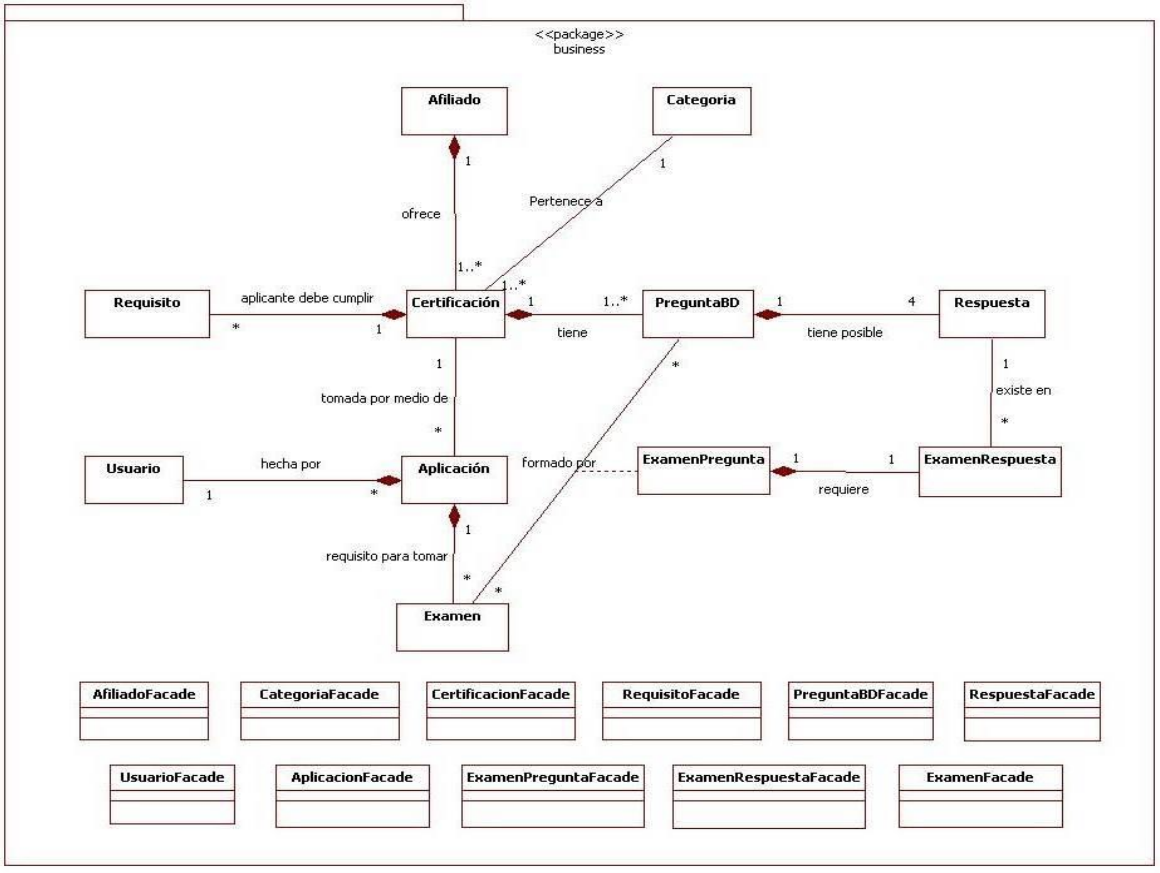
e. AGRUPACIÓN DE CASOS DE USO EN COMPONENTES DE FUNCIONALIDAD ALTAMENTE INDEPENDIENTES

Componente	Descripción	Casos de uso
Catálogo	Componente de la aplicación que implementará todos los casos de uso relacionados con la consulta de las diferentes certificaciones disponibles y el manejo de las aplicaciones en línea	U1, U3, U4, U5, U6, U16
Test	Componente de la aplicación que implementará todos los casos de uso relacionados con la realización de los exámenes de certificación	U7, U8, U9
Cuenta	Componente de la aplicación que implementará todos los casos de uso relacionados con el manejo de la cuenta de usuario	U2, U10
Oficina	Componente de la aplicación que implementará todos los casos de uso relacionados con la creación de afiliados, programas de certificación y el mantenimiento de la base de datos de preguntas y respuestas.	U11
Aplicaciones	Componente de la aplicación que implementará los casos de uso relacionados con la confirmación o rechazo de aplicaciones y el ingreso de cupones de examen	U12, U13, U14, U15

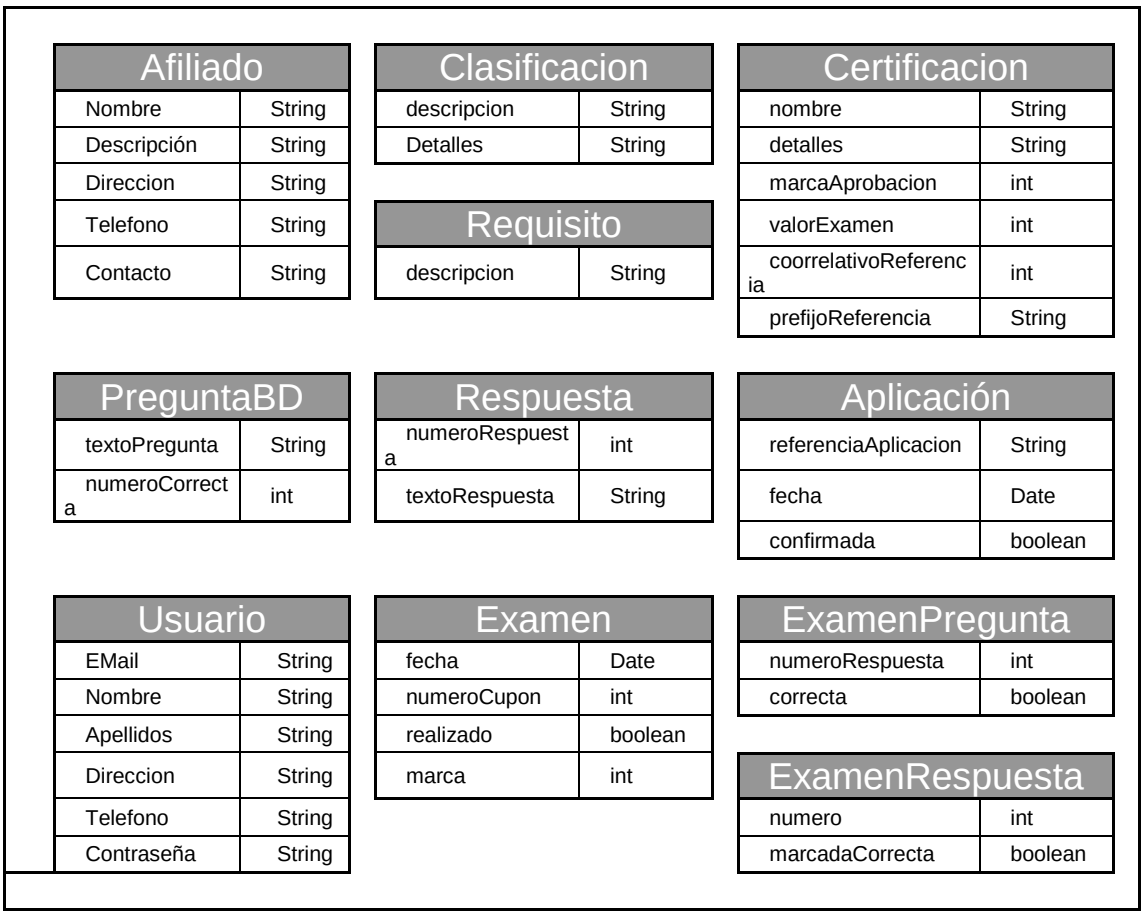
C. ANÁLISIS DEL PROBLEMA

1. ANÁLISIS ESTÁTICO

a. DIAGRAMA DE CLASES



b. LISTA DE ATRIBUTOS DE CLASES



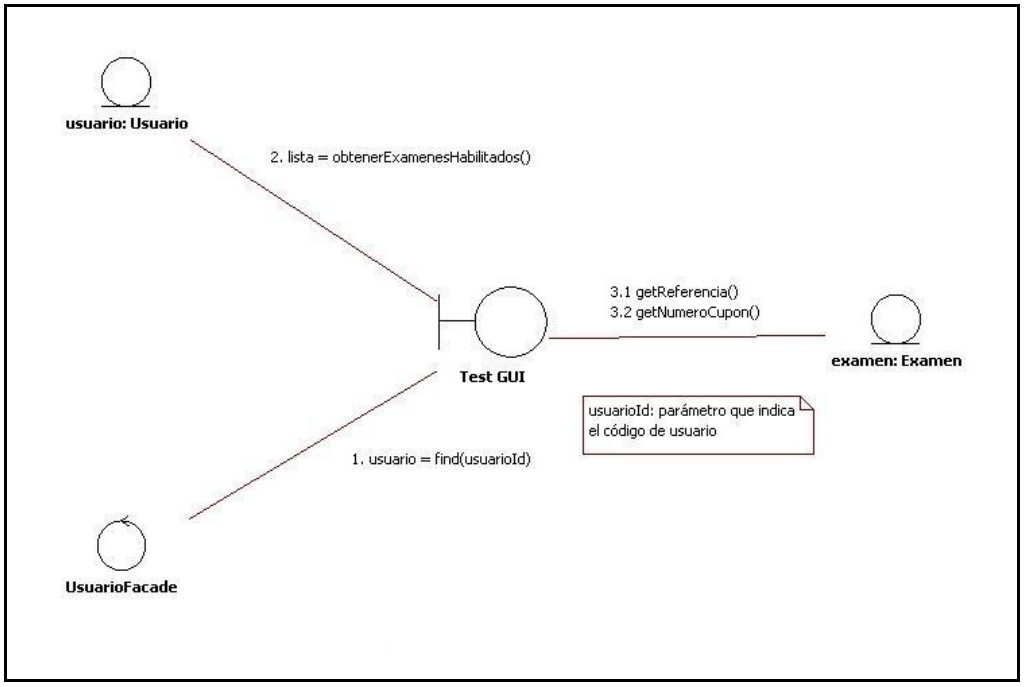
2. ANÁLISIS DINÁMICO.

a. LISTA DE OPERACIONES Y MÉTODOS

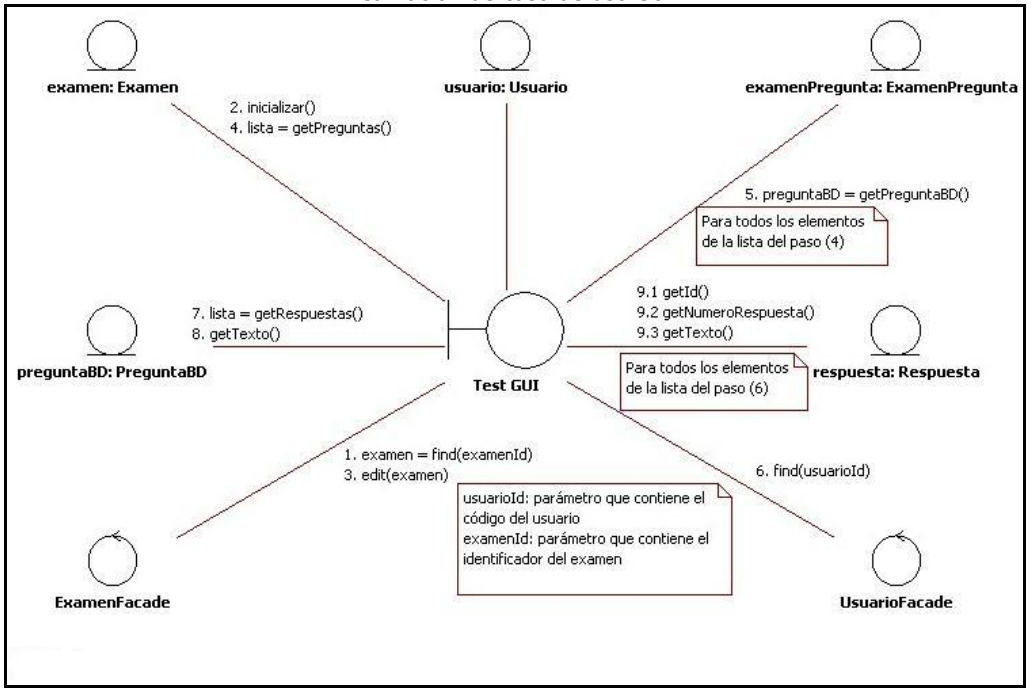
Clase	Método	Descripción	Casos de uso
AplicacionFacade	findByConfirmada(true/false)	Retorna una lista conteniendo todas las aplicaciones confirmadas o no confirmadas	U12,U14
Certificacion	obtenerReferencia()	Retorna el siguiente numero de referencia para una aplicación a la certificación	U6
Certificacion	incrementarCorrelativo()	Incrementa el numero de correlativo de la referencia de la certificación	U6
Examen	inicializar()	Inicializa un examen obteniendo las preguntas de la base de datos	U8
Examen	asignarNota()	Revisa las respuestas proporcionadas con el usuario y las compara con la respuesta correcta de la base de datos de preguntas para asignar la nota del examen	U9
Usuario	existeAplicacion(certificacion)	Verifica la existencia de una aplicación para la certificación pasada como parámetro	U6
Usuario	obtenerExamenesHabilitados()	Retorna una lista conteniendo los exámenes habilitados pendientes de realizar del usuario	U7
UsuarioFacade	findByEmail(email)	Retorna un objeto conteniendo el usuario identificado por la dirección de correo electrónico	U1

b. REALIZACIÓN DE CASOS DE USO PARA EL COMPONENTE DE FUNCIONALIDAD TEST

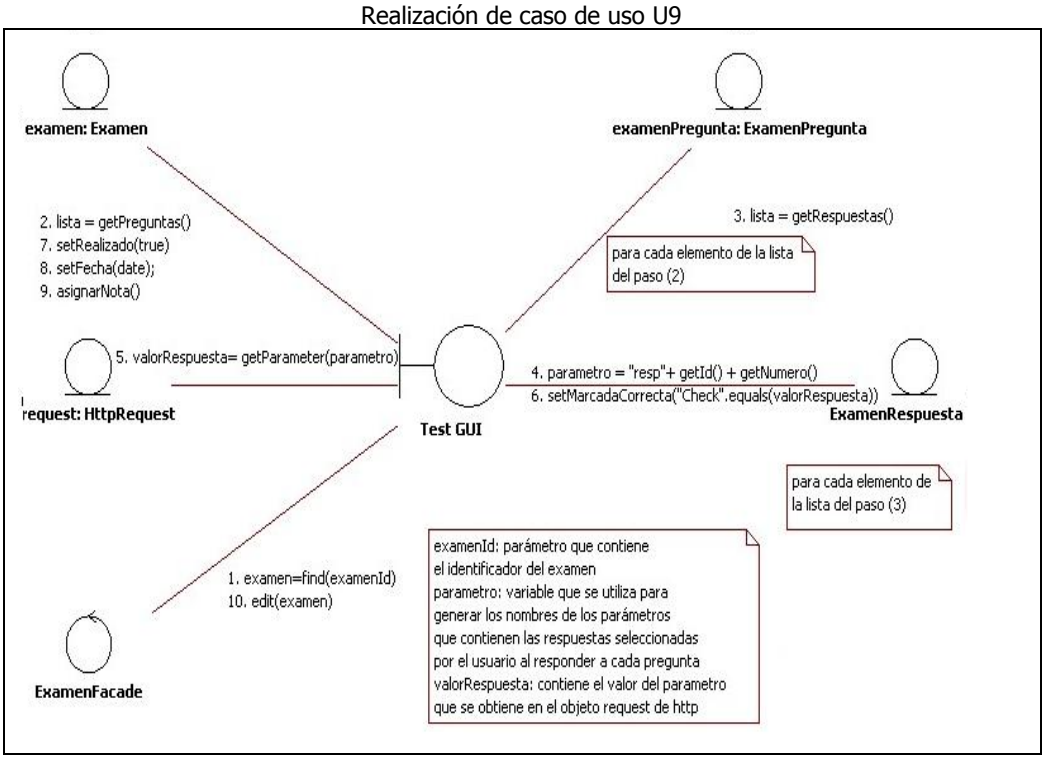
Realización de caso de uso U7



Realización de caso de uso U8



Realización de caso de uso U9



D. DISEÑO DE SUBSISTEMAS

1. SERVLET CONTROLADOR.

Lista de mensajes servlet Test	
Mensaje	**Descripción**
Lista	Se invoca desde el menú horizontal de la página de inicio. Si el usuario ha iniciado sesión en el sistema, retorna la lista de exámenes habilitados para ser realizados.
Realizar	Desde la lista de exámenes habilitados, se selecciona el que se desea realizar en el momento. El sistema inicializa el examen tomando las preguntas de la base de datos y sus posibles respuestas. Retorna una página con la lista de preguntas y posibles respuestas para que el usuario responda cada una de ellas.
Finalizar	Al finalizar el examen, el usuario presiona el botón de finalizar. El sistema verifica las respuestas correctas y asigna la puntuación obtenida. Retorna un mensaje indicando la nota obtenida y la marca de aprobación definida para el examen

2. CLASE DE SERVICIOS DEL NEGOCIO.

Lista de mensajes de la clase TestServerBean	
Mensaje	**Descripción**
listarExamenesHabilitados(id)	Recibe como parámetro el identificador del usuario y retorna la lista de exámenes habilitados para ser realizados en el sistema.
realizarExamen(id)	Recibe como parámetro el identificador del examen que se tomará y realiza tareas de inicialización de las preguntas y respuestas, devolviendo la lista de preguntas con sus respectivas respuestas.
finalizarExamen(id, List respuestas)	Recibe como parámetros el identificador del examen y la lista de respuestas ingresadas por el usuario. Realiza la comparación contra las respuestas correctas de la base de datos y retorna un mensaje con la puntuación obtenida en el examen.

3. COMPONENTES DE LA INTERFAZ DE USUARIO.

Nombre	Tipo	Descripción	Alias	Casos de uso
Lista.jsp	JSP	Página que despliega la lista de exámenes habilitados para realizarse por parte de un usuario con sesión iniciada en el sistema	/Test/lista	U7
Realizar.jsp	JSP	Página que despliega el contenido de un examen de certificación seleccionado en de la lista de exámenes habilitados. Muestra las preguntas con sus posibles respuestas para que el usuario marque las que considere correctas.	/Test/realizar	U8
Notifica.jsp	JSP	Página genérica que se usa para desplegar mensajes del sistema.	/Catalogo/notifica, /Test/notifica, /Cuenta/notifica	U1, U2, U6, U9
Head.jspf	JSP	Fragmento de página JSP que contiene la información que describe el contenido que se incluye en cada página del sistema.	Ninguno	Todos
Header.jspf	JSP	Fragmento de página JSP que contiene el encabezado estándar de las páginas de la aplicación y que se incluye en todas las páginas del sistema.	Ninguno	Todos
Sesion.jspf	JSP	Fragmento de página JSP que contiene el código fuente para desplegar los controles para inicar o cerrar sesión, dependiendo del estado en que se encuentre el usuario.	Ninguno	Todos
Footer.jspf	JSP	Fragmento de página JSP que contiene la información de pie de página estándar de todas las páginas de la aplicación	Ninguno	Todos

4. REALIZACIÓN DE SERVICIOS DEL NEGOCIO PARA LOS CASOS DE USO DEL COMPONENTE TEST.

Diagrama de secuencia del caso de uso U7

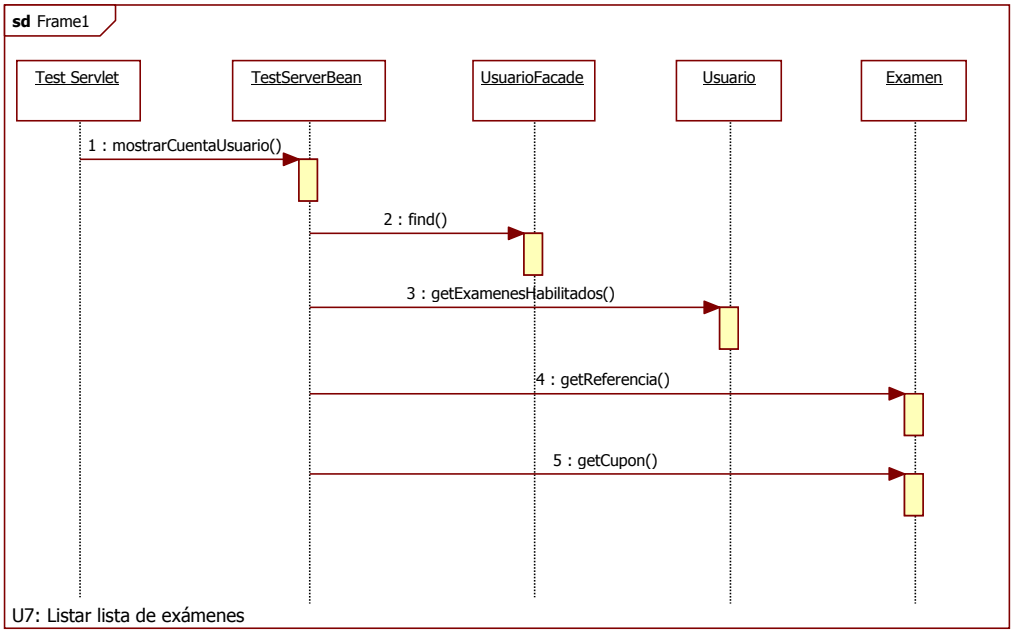


Diagrama de secuencia del caso de uso U8

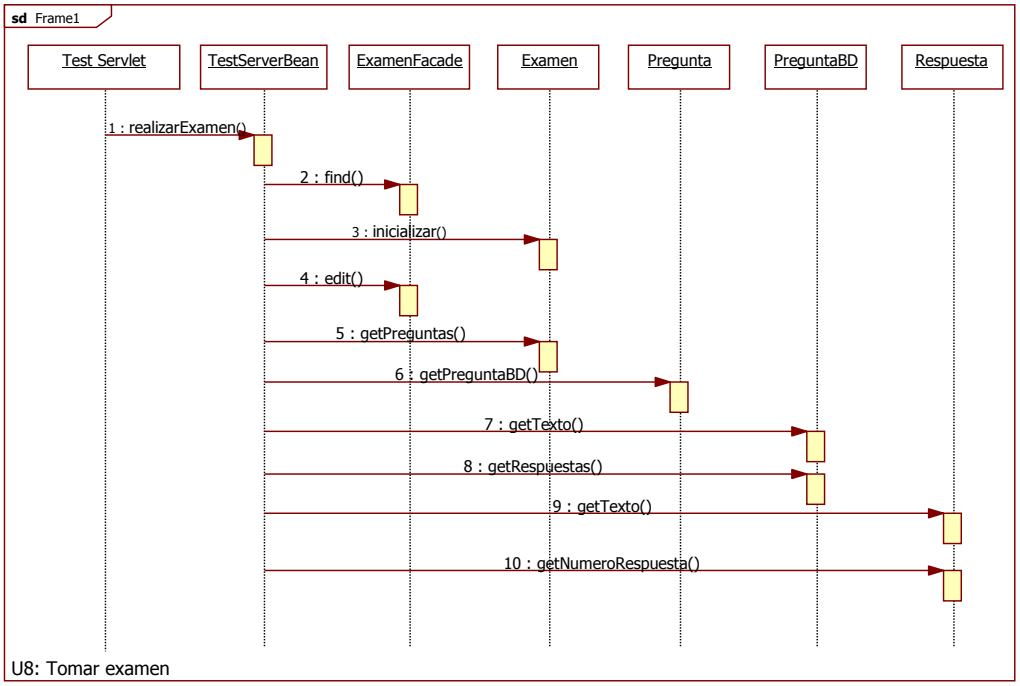
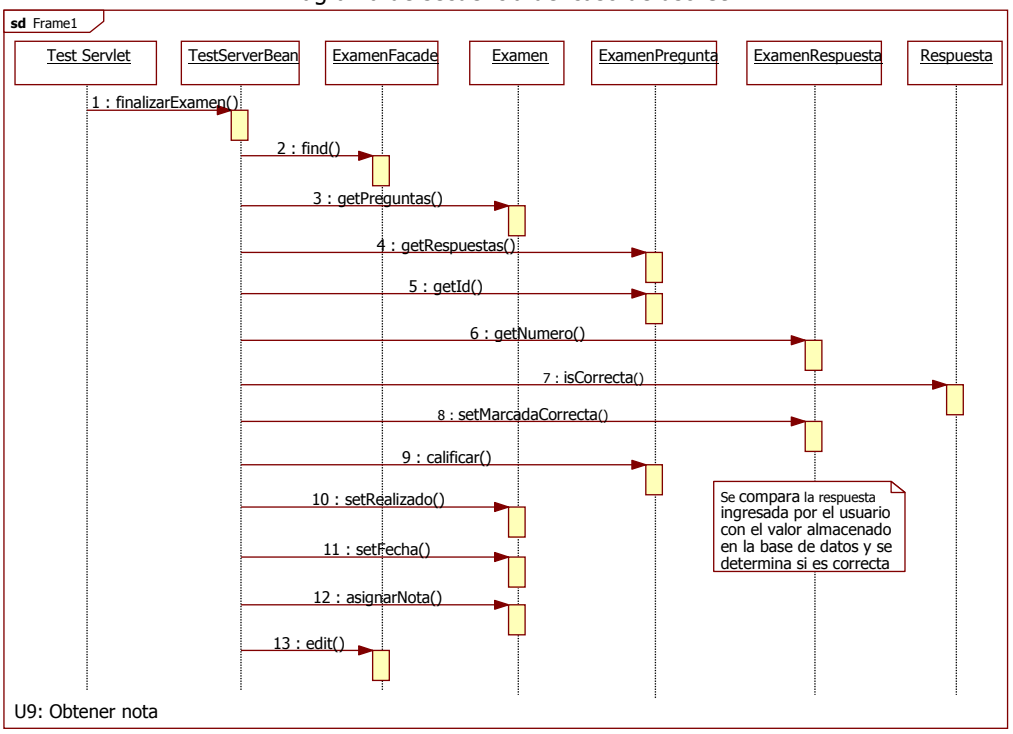
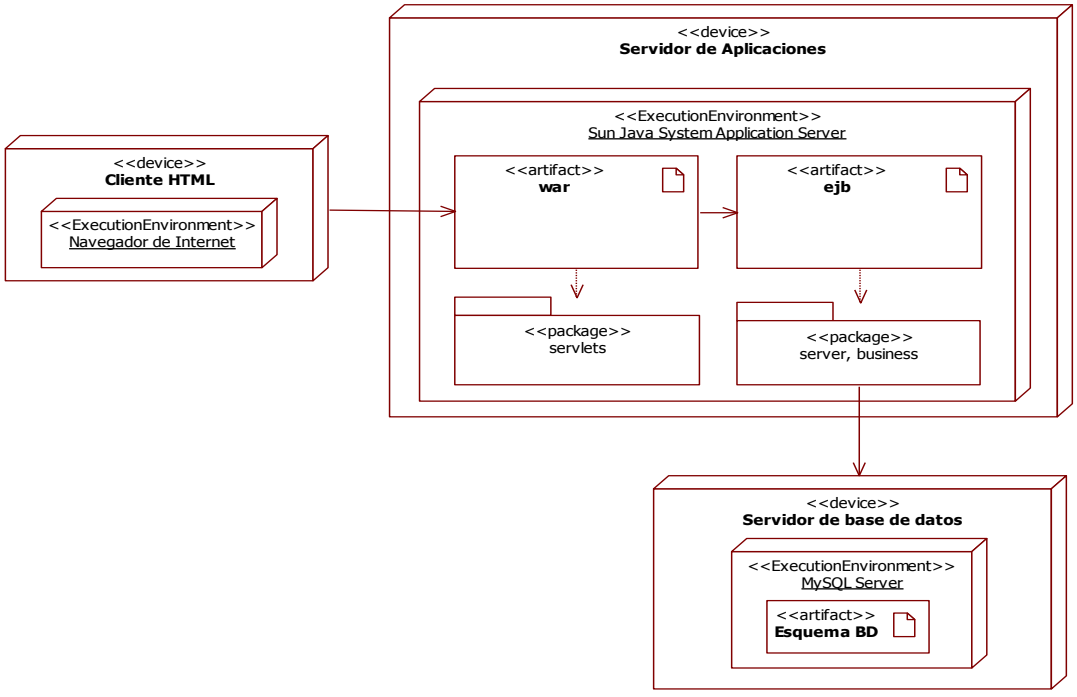


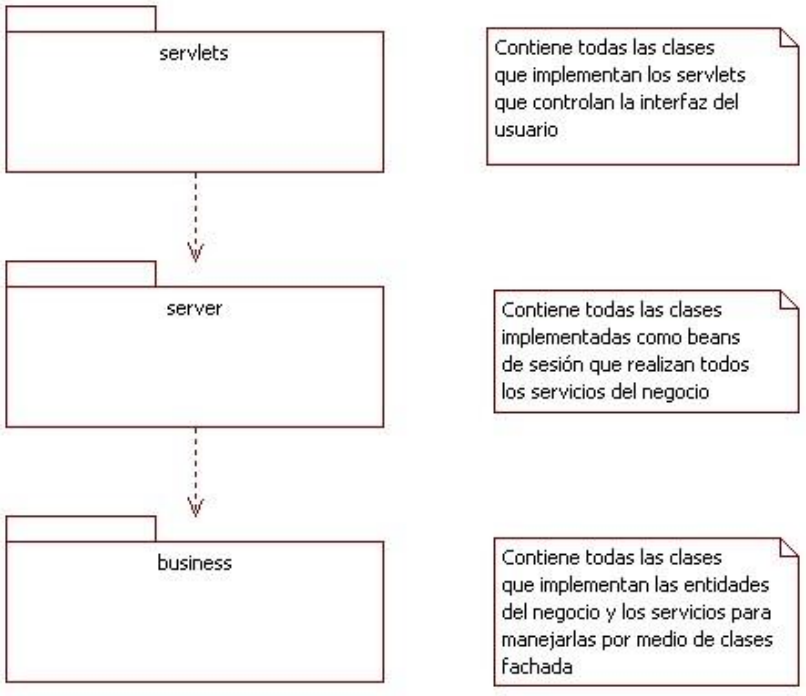
Diagrama de secuencia del caso de uso U9



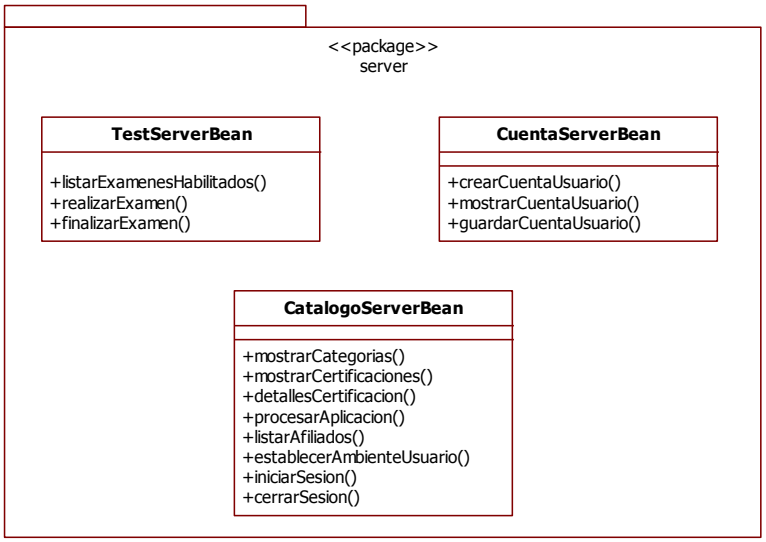
5. DIAGRAMA DE IMPLEMENTACIÓN.



6. DIAGRAMA DE PAQUETES.



7. DIAGRAMA DE CLASES DE REALIZACIÓN DE SERVICIOS DEL NEGOCIO.



E. ESPECIFICACIÓN DE LAS INTERFACES DE CLASES

1. ESPECIFICACIÓN DE LAS INTERFACES PARA EL COMPONENTE DE FUNCIONALIDAD TEST.

a. CLASES DEL DOMINIO DEL NEGOCIO

Especificación de la interfaz de la clase Examen.

Invariantes:

Id obtiene su valor después de la creación en la unidad de persistencia.
Id > 0
referenciaExamen != null
cupon != null

Métodos:

/* Inicializa el examen con las preguntas de la base de datos, justo antes de ser tomado */

public void inicializar()

Precondiciones:

realizado == false
preguntas.isEmpty() == true

Poscondiciones:

preguntas.isEmpty() == false

/* Asignar nota al examen */
Public void finalizarExamen()

Precondiciones:

realizado == true
preguntas.isEmpty() == false

Poscondiciones:

Marca > 0
realizado == trae

b. CLASES DE SERVICIOS DEL NEGOCIO

Especificación de la clase TestServerBean

Invariantes: No tiene atributos

Métodos:

public void inicializarExamen()

Precondiciones:

usuarioFacade.find(usuarioId) != null
usuario.isSesionActiva() == true
examen=examenFacade.find(examenId) != null

Poscondiciones:

El examen es inicializado con las preguntas de la base de datos y está listo para ser respondido por el usuario
Public void finalizarExamen()

Precondiciones:

usuarioFacade.find(usuarioId) != null
usuario.isSesionActiva() == true
examen=examenFacade.find(examenId) != null

Poscondiciones:

marca != null

F. DESARROLLO DE LA APLICACIÓN

En esta sección se ilustran los pasos a seguir en la etapa de desarrollo del sistema. La metodología empleada se basa en el marco propuesto, al permitir que los componentes de funcionalidad altamente independientes se puedan desarrollar simultáneamente.

1. PLANIFICACIÓN DEL INCREMENTO EN PROCESO. A continuación se presenta la lista de los componentes a desarrollar en el incremento en proceso:

- Entidades de las clases del dominio del negocio
- Clases fachada para las entidades del dominio del negocio
- Clases del componente de funcionalidad Test

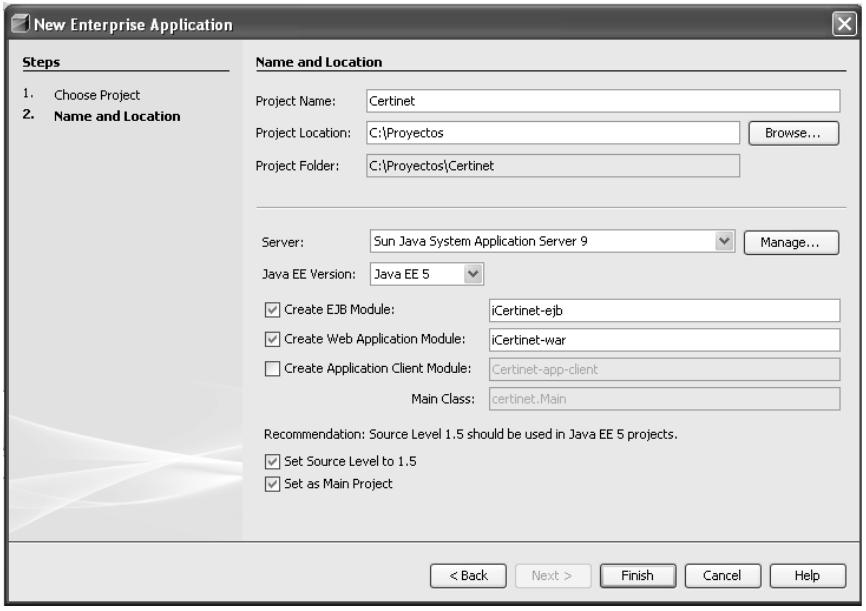
2. CREACIÓN DE UN PROYECTO DE APLICACIÓN EMPRESARIAL. En NetBeans 5.5, una aplicación empresarial se crea a partir de un proyecto que hace referencia a tres tipos diferentes de proyecto que cumplen con una función específica dentro del sistema.

Proyecto de módulo Web. Contiene todos los componentes para crear una aplicación Web, tales como páginas JSP, Servlets e interfaces de usuario JSF.

Proyecto de módulo EJB. Contiene todos los componentes de la aplicación que se desarrollarán como Enterprise Java Beans y entidades manejadas por la API de persistencia.

Proyecto de módulo de aplicación cliente. Contiene los componentes de la aplicación que se desarrollarán como aplicaciones de consola usando JFC y librerías de clases como Swing.

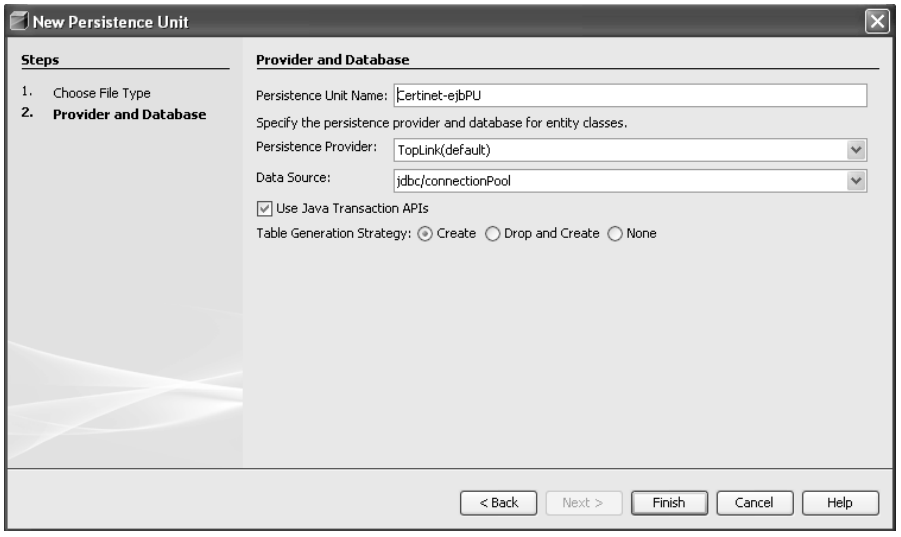
Creación de un proyecto de aplicación empresarial en NetBeans



La aplicación Certinet contiene componentes Web y EJB, por lo que se crean los proyectos para dichos módulos especificando la versión del kit de desarrollo de Java a utilizar y el servidor de aplicaciones sobre el que se implementará la solución. La ilustración anterior muestra el cuadro de diálogo proveído por el entorno de desarrollo para la creación del proyecto.

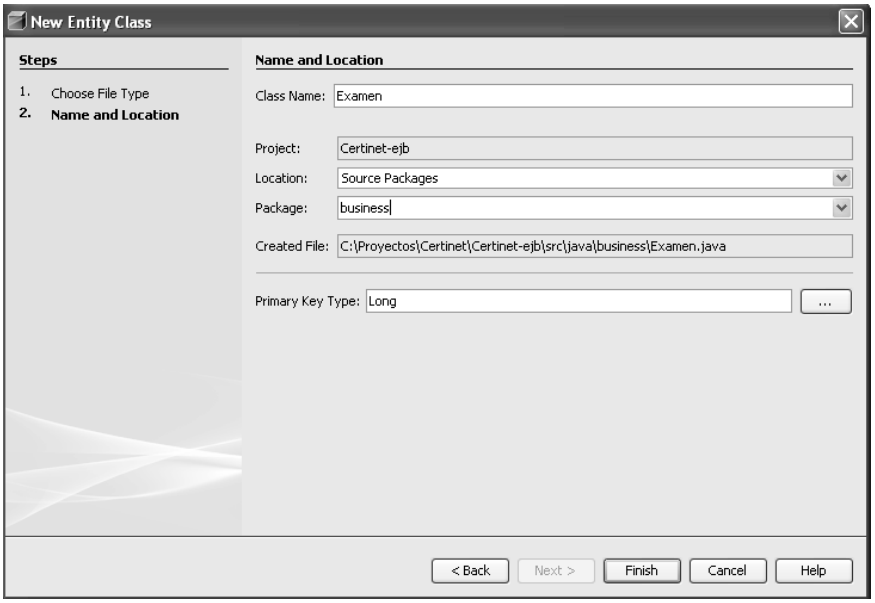
a. CREACIÓN DE LA UNIDAD DE PERSISTENCIA. La API de persistencia de la especificación EJB 3.0 requiere que exista un archivo llamado "persistence.xml", cuyo contenido consiste en descriptores referentes a la conexión hacia el servidor de base de datos que el servidor de aplicaciones usará para tener acceso a la información. La unidad de persistencia se debe incluir en el proyecto del módulo EJB.

Creación de la unidad de persistencia en NetBeans



3. DESARROLLO DE CLASES DE ENTIDADES DEL DOMINIO DEL NEGOCIO. Las clases del dominio del negocio se desarrollan como clases de entidad dentro del proyecto del módulo EJB, creándolas con características de Java Beans y agregando las anotaciones de la API de persistencia. NetBeans facilita esta tarea por medio de una funcionalidad que permite encapsular los métodos de las clases especificando acceso privado para todos sus atributos y creando los métodos de acceso para cada uno de ellos. La siguiente figura muestra el cuadro de dialogo de NetBeans para agregar una nueva clase de entidad.

Creación de una clase de entidad en NetBeans

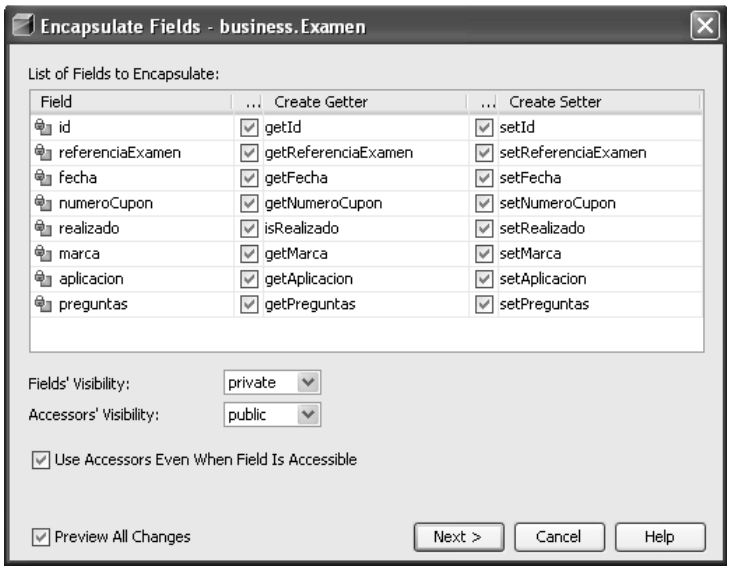


Una vez finalizado el cuadro de diálogo, NetBeans crea una clase básica derivada de una plantilla, la cual deberá ser completada con todos los atributos de la clase del dominio del negocio, incluyendo los atributos derivados de objetos de las otras clases con las que tiene relación y los métodos adicionales a los de acceso de cada atributo.

```
package business;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
/**
 * Entity class Examen
  */
@Entity
public class Examen implements Serializable {
    @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   /** Creates a new instance of Examen */
     public Examen() {
   }
   /**
    * Gets the id of this Examen.
    */
   public Long getId() {
       return this.id;
   }
   /**
    * Sets the id of this Examen to the specified value.
    */
   public void setId(Long id) {
       this.id = id;
   }
 }
```

Al completar la clase con sus atributos y métodos, se procede a encapsularla para convertirla en un Java Bean. Esto se logra seleccionando la opción Encapsulate Fields del menú Refactor.

Encapsulamiento de una clase en NetBeans



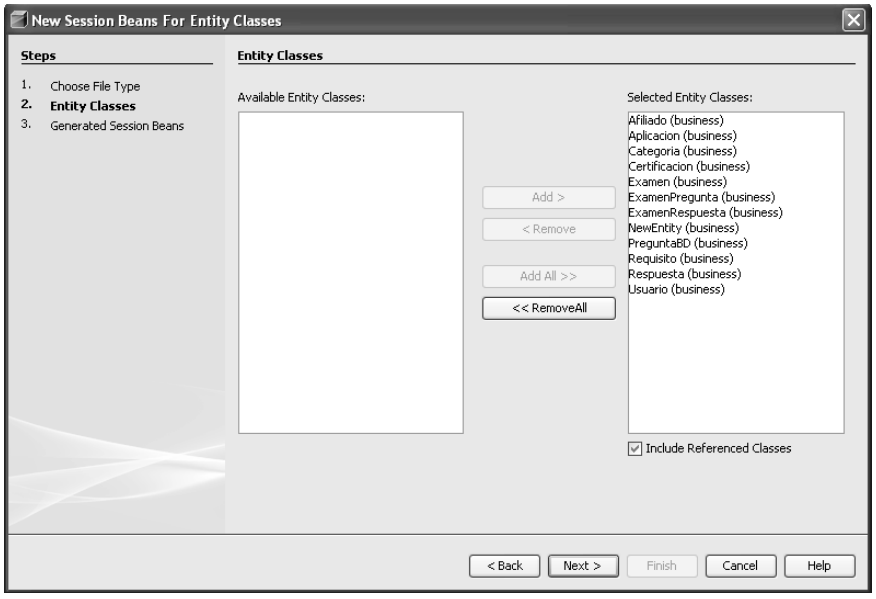
Al finalizar esta tarea, el desarrollo de la clase está concluido y está lista para el siguiente paso que consiste en crear su clase fachada.

El procedimiento descrito con anterioridad se aplica a todas las clases del dominio del negocio para crearlas en el proyecto como clases de entidad manejadas por la unidad de persistencia.

4. DESARROLLO DE CLASES FACHADA. Similar a la forma como se desarrollan las clases de entidad, NetBeans posee plantillas para la creación de clases fachada a partir de éstas. Las clases fachada se incluyen en el proyecto del módulo EJB, dentro del mismo paquete que las clases de entidad.

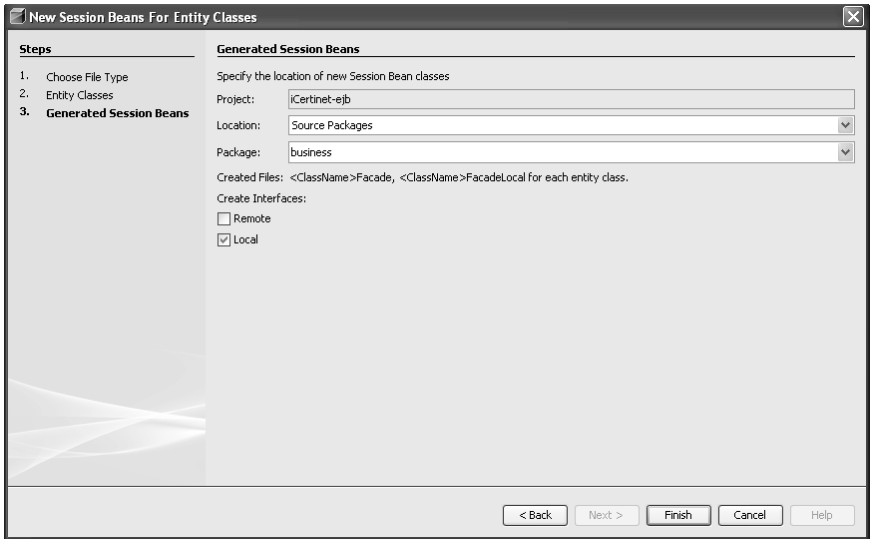
La figura muestra el cuadro de diálogo de la herramienta en la que se especifican las clases de entidad a las que se les generará su clase fachada.

Creación de clases fachada en NetBeans



En la siguiente parte del cuadro de diálogo se ingresan la ubicación donde se guardarán las clases fachada y el paquete que las contendrá.

Ubicación de clases fachada en NetBeans



5. DESARROLLO DEL COMPONENTE DE FUNCIONALIDAD TEST. Los componentes de funcionalidad del sistema se desarrollan de acuerdo a la prioridad

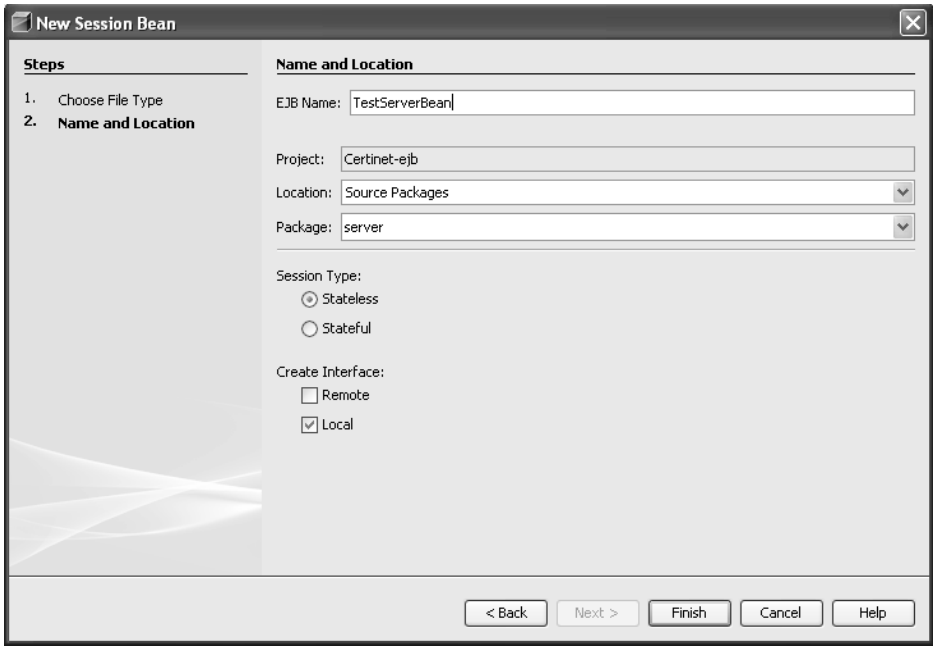
establecida en la planificación del incremento en proceso siguiendo los pasos que se detallan a continuación:

1. Desarrollo de la clase de servicios del negocio del componente
2. Desarrollo de la clase del servlet controlador
3. Desarrollo de los componentes de implementación de la interfaz de usuario

a. DESARROLLO DE CLASES DE SERVICIOS DEL NEGOCIO. Las clases de servicios del negocio se desarrollan como EJB de sesión dentro del proyecto del módulo EJB y forman parte del paquete llamado server.

La figura muestra el cuadro de diálogo para agregar la clase de realización de servicios del negocio del componente Test.

Creación de clase de servicios del negocio en NetBeans



A continuación se muestra el código fuente de la clase después de haber agregado los mensajes que realizan los servicios del negocio. Las anotaciones @EJB indican las referencias hacia los EJB que implementan las clases fachada de las entidades.

```
package server;
import business.Examen;
import business.ExamenFacadeLocal;
import business.ExamenPregunta;
import business.ExamenPreguntaFacadeLocal;
import business.ExamenRespuesta;
import business.ExamenRespuestaFacadeLocal;
import business.Usuario;
import business.UsuarioFacadeLocal;
import java.util.Date;
import java.util.Iterator;
import javax.ejb.EJB;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
/**
 *
 * @author Sergio Chong
 */

@Stateless
public class TestServerBean implements TestServerLocal {

    /* Referencia hacia los bean de las clases fachada */
    @EJB
    private ExamenPreguntaFacadeLocal examenPreguntaFacade;

    @EJB
    private ExamenRespuestaFacadeLocal examenRespuestaFacade;

    @EJB
    private ExamenFacadeLocal examenFacade;

    @EJB
    private UsuarioFacadeLocal usuarioFacade;

    /** Creates a new instance of TestServerBean */
    public TestServerBean() {
    }

    /* Implementacion de los mensajes de realización de servicios del negocio */

    public void listarExamenesHabilitados(HttpServletRequest request, HttpServletResponse response) {
        HttpSession sesion = request.getSession(false);
        Long usuarioId = (Long) sesion.getAttribute("usuarioId");
        Usuario usuario = usuarioFacade.find(usuarioId);
        request.setAttribute("usuario", usuario);
    }

    public void realizarExamen(HttpServletRequest request, HttpServletResponse response){
        HttpSession sesion = request.getSession(false);
        Long usuarioId = (Long) sesion.getAttribute("usuarioId");
        Long examenId = Long.parseLong(request.getParameter("exid"));
        Examen examen = examenFacade.find(examenId);
        examen.inicializar();
        em.merge(examen);
        sesion.setAttribute("examenId", examenId);
        request.setAttribute("examen", examen);
        Usuario usuario = usuarioFacade.find(usuarioId);
        request.setAttribute("usuario", usuario);
    }

    public void finalizarExamen(HttpServletRequest request, HttpServletResponse response) {
        HttpSession sesion = request.getSession(false);
        Long usuarioId = (Long) sesion.getAttribute("usuarioId");
        Long examenId = (Long) sesion.getAttribute("examenId");
        Examen examen = examenFacade.find(examenId);
```

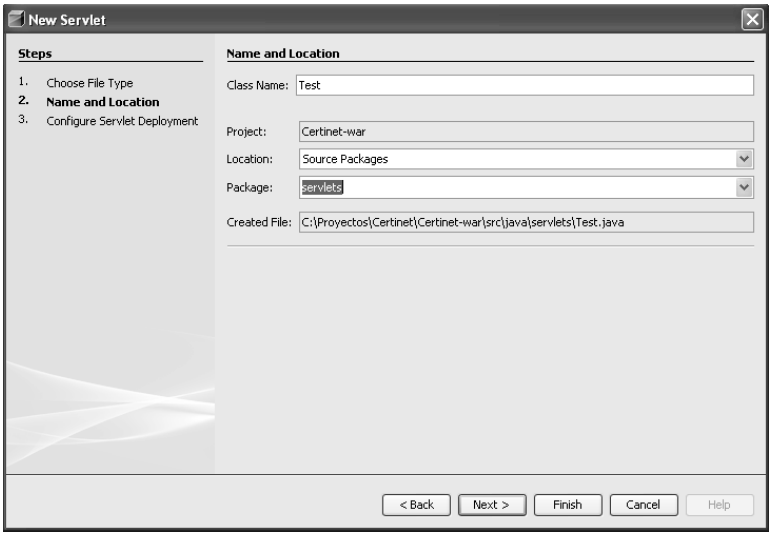
```
        Date date = new Date();
        for (Iterator it = examen.getPreguntas().iterator(); it.hasNext();) {
           ExamenPregunta examenPregunta = (ExamenPregunta) it.next();
           for (Iterator ite = examenPregunta.getRespuestas().iterator(); ite.hasNext();) {
              ExamenRespuesta examenRespuesta = (ExamenRespuesta) ite.next();
              String parametro = "resp"+  Long.toString(examenPregunta.getId()) + "_" +
                                  Integer.toString(examenRespuesta.getNumero());
              String valorRespuesta = request.getParameter(parametro);
              examenRespuesta.setMarcadaCorrecta("Check".equals(valorRespuesta));
              em.merge(examenRespuesta);
           }
           examenPregunta.calificar();
           em.merge(examenPregunta);
        }

        examen.setRealizado(true);
        examen.setFecha(date);
        examen.asignarNota();
        em.merge(examen);
        String mensaje = "Nota obtenida: " + Integer.toString(examen.getMarca()) +
                              "Marca de aprobación: " +
                               Integer.toString(examen.getAplicacion().getCertificacion().getMarcaAprobacion());
        request.setAttribute("mensaje",mensaje);
     }
  }
```

b. DESARROLLO DEL SERVLET DEL COMPONENTE DE FUNCIONALIDAD TEST. Las clases servlet se desarrollan dentro del proyecto del módulo web y hacen referencia a los EJB de realización de servicios del negocio para enviarles los mensajes de cada caso de uso en particular. Estas clases cumplen con la función de controlador, por lo que su funcionamiento se lleva a cabo de la siguiente manera:

1. Recepción del mensaje proveniente de la petición del cliente.
2. Identificación del caso de uso requerido por el cliente.
3. Envío del mensaje hacia el EJB de realización de servicios del negocio respectivo y recepción de la información retornada por éste.
4. Direccionamiento de la salida hacia el componente que implementa la interfaz del usuario.

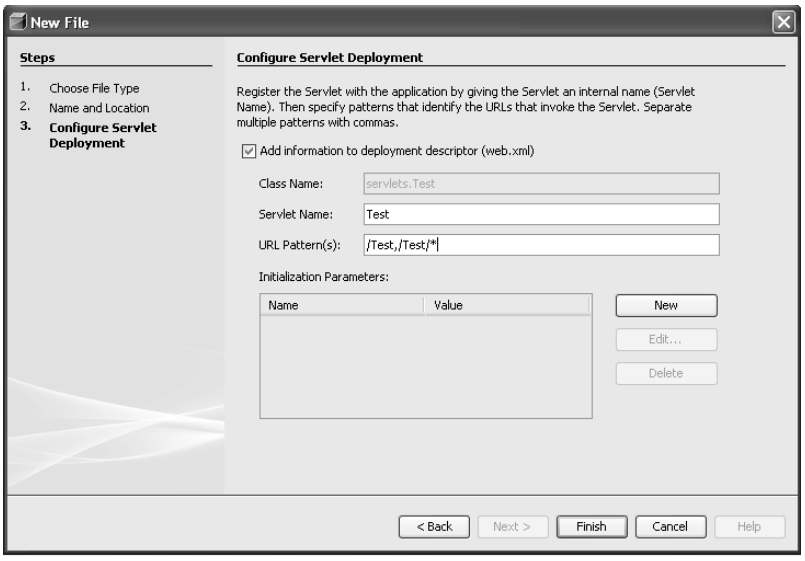
El cuadro de diálogo que se muestra a continuación corresponde a la creación del servlet del componente de funcionalidad Test en el proyecto del módulo Web de la aplicación.

Creación de un servlet en NetBeans



Para la recepción de los mensajes que recibe el servlet, se adoptó la modalidad de envío por medio de la dirección URL, en la que el servlet Test recibe el mensaje "lista" al ser invocado en el navegador de Internet agregando la terminación "/lista" a la dirección, por lo que deberá crearse un alias que indique que el servlet se puede invocar en el patrón "/Test/*". El alias se crea en el cuadro de dialogo siguiente:

Configuración de un servlet en NetBeans



El servlet debe ser completamente independiente de la implementación de las reglas y servicios del negocio, por lo que únicamente debe contener el código fuente

necesario para implementar las cuatro funciones enumeradas con anterioridad, tal y como se muestra a continuación:

```
package servlets;
import java.io.*;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import javax.ejb.EJB;
import javax.servlet.*;
import javax.servlet.http.*;
import server.TestServerLocal;

public class Test extends HttpServlet {

   /* Referencia hacia el bean de sesion que implementa la realización de servicios del negocio */
   @EJB
   private TestServerLocal testServerBean;

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
    * @param request servlet request
    * @param response servlet response
    */

   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      HttpSession sesion = request.getSession(false);
      try {
         ArrayList data = null;
         RequestDispatcher dispatcher;
         ServletContext context = getServletContext();

         /* Obtención del mensaje enviado como parte de la dirección URL */
         String name = request.getPathInfo();
         name = name.substring(1);

         /* Condiciones para el direccionamiento de la petición hacia el método que implementa
            El caso de uso en el bean de sesion */

         if ("lista".equals(name)) {
            if (sesion == null) {
               request.setAttribute("mensaje", "Opción restringida para usuarios registrados");
               name = "notifica";
            } else {
               testServerBean.listarExamenesHabilitados(request, response);
            }
         } else if ("realizar".equals(name)) {
            testServerBean.realizarExamen(request, response);
         } else if ("finalizar".equals(name)) {
            testServerBean.finalizarExamen(request, response);
            name = "notifica";
         }

         /* Direccionamiento de la salida hacia el componente JSP de la capa de presentacion */
         dispatcher = context.getNamedDispatcher(name);
         if (dispatcher == null) {
            dispatcher = context.getNamedDispatcher("Error");
         }
         dispatcher.forward(request, response);
      } catch (Exception e) {
         log("Excepcion en Catalog.doGet()");
      }
      out.close();
   }
```

```
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */

    public String getServletInfo() {
        return "Short description";
    }

    // </editor-fold>
}
```

c. DESARROLLO DE LA INTERFAZ DE USUARIO. Los componentes de la interfaz de usuario para el componente de funcionalidad Test se implementan por medio de JavaServer Pages.

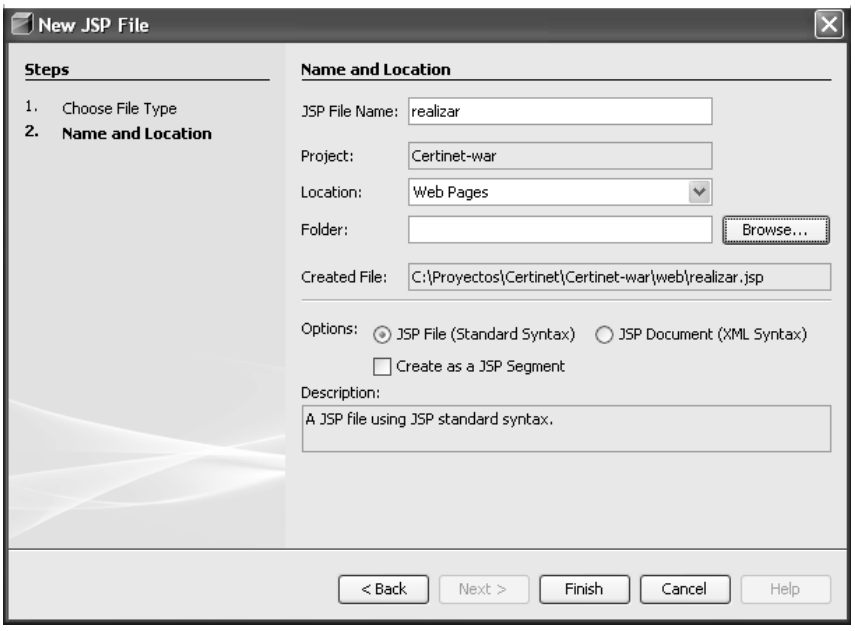
La plantilla de diseño de la interfaz de usuario fue tomada de un recurso de código abierto (http://www.oswd.org), en el que se pueden obtener plantillas con diferentes diseños, entre los que se puede escoger el que más se adapte a los requerimientos de la aplicación, de tal forma que se obtenga un buen componente sin costos de adquisición y con la posibilidad de hacer modificaciones a su código fuente para convertirlo en una página JSP.

Con el objetivo de desarrollar una interfaz de usuario fácilmente mantenible, se crearon tres fragmentos de página para evitar redundar en partes repetitivas como el encabezado, pie de página y control de la información de despliegue dependiendo del estado de inicio de sesión del usuario, los cuales se incluyen en todas las páginas por medio de directivas de inclusión.

Las páginas JSP se desarrollan como parte del proyecto del módulo Web, en cuyo contenido se hace referencia a los parámetros enviados por el servlet cuando la salida se direcciona hacia ellas.

La siguiente figura muestra el cuadro de diálogo desplegado por la herramienta para agregar la página JSP que presenta la interfaz de usuario para responder las preguntas del examen.

Creación de un JSP en NetBeans



A continuación se muestra el código fuente del archivo realizar.jsp después de haber agregado la programación necesaria para mostrar la información.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%--
The taglib directive below imports the JSTL library. If you uncomment it,
you must also add the JSTL library to the project. The Add Library... action
on Libraries node in Projects view can be used to add the JSTL 1.1 library.
--%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="sp">
   <!-- AQUI VA EL PRIMER INCLUDE HEAD -->
   <%@ include file="WEB-INF/head.jspf" %>
   <body>
      <%@ page import = "java.io.*" %>
      <%@ page import = "java.util.Iterator" %>
```

```
<%@ page import = "java.util.List" %>
<%@ page import = "business.Usuario" %>
<%@ page import = "business.Aplicacion"%>
<%@ page import = "java.util.Collection" %>
<%@ page import = "business.Examen" %>
<%@ page import = "business.ExamenPregunta" %>
<%@ page import = "business.PreguntaBD" %>
<%@ page import = "business.Respuesta" %>

<!-- wrap starts here -->
<div id="wrap">

   <%@ include file="WEB-INF/header.jspf" %>
   <!-- content-wrap starts here -->
   <div id="content-wrap">
      <div id="main">

         <a name="TemplateInfo"></a>
         <h1>Preguntas</h1>
         <form method="post" id="search" action="finalizar">
            <%
            Usuario usuario = (Usuario) request.getAttribute("usuario");
            String nombreCompleto = usuario.getNombres() + " " + usuario.getApellidos();
            Examen examen = (Examen) request.getAttribute("examen");
            int numeroPregunta = 0;
            Collection <ExamenPregunta> examenPreguntas = examen.getPreguntas();
            for (Iterator it = examenPreguntas.iterator(); it.hasNext();) {
               numeroPregunta++;
               ExamenPregunta examenPregunta = (ExamenPregunta) it.next();
               PreguntaBD preguntaBD = examenPregunta.getPreguntaBD();
            %>
            <p><strong><%=numeroPregunta%>. <%=preguntaBD.getTexto()%></strong></p>
            <p>
            <%
            for (Iterator ite = preguntaBD.getRespuestas().iterator(); ite.hasNext();) {
                  Respuesta respuesta = (Respuesta) ite.next();
            %>
            <p>
               <input type=checkbox name="resp<%=examenPregunta.getId()%>_<%=respuesta.getNumeroRespuesta()%>"
                  value="Check">
            <%=respuesta.getTexto()%> </p>
            <%
            }
            %>
            <BR> </BR>
            <%
            }
            %>
            <p><input name="iniciar" class="iniciarbutton" value="Finalizar Examen" type="submit" /></p>
         </form>
          </div>
      <div id="sidebar">
         <h1>Examen a realizar</h1>
         <ul class="sidemenu">
            <%
            Collection <Aplicacion> aplicaciones = usuario.getAplicaciones();
            if (aplicaciones.isEmpty()) {
            %>
            <strong>No hay aplicaciones</strong>
            <%
            } else {
               for (Iterator it = aplicaciones.iterator(); it.hasNext();) {
                  Aplicacion aplicacion = (Aplicacion) it.next();
                  Collection <Examen> examenes = aplicacion.getExamenes();
                  for (Iterator ite = examenes.iterator(); ite.hasNext();) {
                     Examen examena = (Examen) ite.next();
                     if ((examena.getNumeroCupon() != null) && (!examena.isRealizado()))
            %>
            <li><a  href="realizar?exid=<%=examena.getId()%>"><%=aplicacion.getCertificacion().getNombre()+":
                        "+examena.getReferenciaExamen() + " Cupón " + examena.getNumeroCupon()%></a></li>
```

```
                <%
                      }
                    }
                  }
                  %>
                </ul>
              </div>
              <!-- content-wrap ends here -->
            </div>
            <%@ include file = "WEB-INF/footer.jspf" %>
            <!-- wrap ends here -->
          </div>
        </body>
      </html>
```

G. PRUEBA

1. INSPECCIÓN DEL CÓDIGO FUENTE. La inspección del código fuente consiste en validar que éste cumpla con las especificaciones del diseño de los subsistemas y que la invocación de los mensajes entre objetos se haga tal y como se documentó en los diagramas de secuencia para la realización de los casos de uso.

a. INSPECCIÓN DE CÓDIGO FUENTE DE ENTIDADES. Consiste en la validación de que el código fuente desarrollado cumpla con las especificaciones de diseño de las clases. La validación se registra por medio de la lista de verificación que se muestra a continuación.

Entidad: Examen

Atributos			
Atributo	**Tipo**	**Acceso Privado**	**Métodos de acceso**
referenciaExamen	String	Si	Si
Fecha	Date	Si	Si
numeroCupon	String	Si	Si
Realizado	Boolean	Si	Si
Marca	Int	Si	Si
Métodos y operaciones			
Método	**Valor de retorno**	**Parámetros**	**Tipo de acceso**
inicializar()	void	ninguno	Público
asignarNota()	void	ninguno	Público
Relaciones			
Relación	**Anotación**	**Objeto**	**Métodos de acceso**
Aplicación	ManyToOne	Si	Si
ExamenPregunta	OneToMany	Si	Si

b. INSPECCIÓN DE CÓDIGO FUENTE DE CLASES DE SERVICIOS DEL NEGOCIO. Consiste en la validación de que el código fuente de las clases de servicios del negocio cumpla con las especificaciones del diseño y que los casos de uso se realicen en forma correcta. Se hace por medio de una lista de verificación de los mensajes de la clase y los mensajes que envía hacia los objetos involucrados en la realización de cada caso de uso.

Clase: TestServerBean

Mensaje: listarExamenesHabilitados() Caso de uso: U7		
	usuario=usuarioFacade.find(examenId)	si
	lista = usuario.obtenerExamenesHabilitados()	si
	examen.getReferencia()	si
	examen.getNumeroCupon()	si
	examen.getId()	si

Mensaje: realizarExamen() Caso de uso: U8		
	examen=examenFacade.find(examenId)	si
	examen.inicializar()	si
	examenFacade.edit(examen)	si
	lista = examen.getPreguntas()	si
	PreguntaBD = pregunta.getPreguntaBD()	si
	lista = preguntaBD.getRespuestas()	si
	PreguntaBD.getTexto()	si
	respuesta.getId()	si
	respuesta.getNumeroRespuesta()	si
	respuesta.getTexto()	si

Mensaje: finalizarExamen() Caso de uso U9		
	examen = examenFacade.find(examenId)	si
	lista = examen.getPreguntas()	si
	Respuestas = examenPregunta.getRespuestas()	si
	parametro = examenRespuesta.getId() + "_" + examenRespuesta.getNumero()	si
	examenRespuesta.setMarcadaCorrecta()	si
	examen.setRealizado()	si
	examen.setFecha()	si
	examen.asignarNota()	si
	examenFacade.edit()	si

c. INSPECCIÓN DE CÓDIGO FUENTE DE SERVLETS. Consiste en la validación de que el código fuente de las clases servlet cumpla con las especificaciones del diseño y que direccione la salida hacia la interfaz correcta.

Mensaje	Método	Salida	Correcto
Lista	listarExamenesHabilitados()	lista.jsp	si
Realizar	realizarExamen()	realizar.jsp	si
Finalizar	finalizarExamen()	notifica.jsp	si

2. PRUEBAS FUNCIONALES. La prueba funcional consiste en la selección de un conjunto de datos relacionados que se cargan al sistema para realizar corridas de prueba y verificar que los resultados obtenidos sean correctos.

a. CASO DE PRUEBA.

Examen de certificación en ajuste para el rendimiento de bases de datos.
Contenido: 10 preguntas
Marca de aprobación: 70% de preguntas correctas

1. Ciertos tipos de parámetros del sistema operativo son configurables por el DBA, ¿cómo se les llama a dichos parámetros?

    a. Parámetros del kernel
    b. Parámetros base
    c. Parámetros de inicialización
    d. Parámetros elementales

2. ¿Qué tipo de estructuras son protegidas por el mecanismo de los Latches?

    a. Archivos de datos
    b. Archivos de control
    c. Tablas
    d. Estructuras de memoria

3. El DBA desea asegurarse de que ciertos usuarios tengan acceso a todos los recursos del servidor sin restricción. ¿En qué grupo debe asignarlos?

    a. OTHER
    b. SYS
    c. LOW
    d. CONSUMER

4. En la configuración predeterminada de la base de datos, ¿cuál es el método usado para las conexiones de los clientes?

    a. Servidor dedicado
    b. Servidor compartido
    c. Cliente/Servidor
    d. ODBC

5. ¿Cuántos planes de recursos de la base de datos pueden estar activos simultáneamente?

    a. Todos los existentes
    b. Ninguno
    c. Uno
    d. Hasta cuatro

6. ¿Cuál de los siguientes comandos no causa operaciones de ordenamiento?

    a. Order by
    b. Analyze table
    c. Select distinct
    d. Todos causan operaciones de ordenamiento

7. Cuando un interbloqueo ocurre, ¿qué sesion será cancelada para resolverlo?
    a. La sesión que causa el interbloqueo
    b. La sesión que detecta el interbloqueo
    c. La sesión con menor cantidad de trabajo
    d. La sesión más reciente

8. ¿Bajo qué porcentaje de utilización se recomienda mantener el CPU para obtener un rendimiento óptimo?
    a. 70
    b. 80
    c. 90
    d. 100

9. ¿Cuál de las siguientes herramientas es la más adecuada para monitorear los bloqueos?
    a. Visor de sesiones
    b. Monitor de bloqueos
    c. Traza de datos
    d. Analizador SQL

10. ¿Cuál de los siguientes no es parte del mecanismo de registro de transacciones?
    a. Escritor del log
    b. Escritor de la base de datos
    c. Puntos de verificación
    d. Archivado del registro

b. EJECUCIÓN DEL CASO DE PRUEBA. La siguiente figura muestra la página desplegada como producto de haber seleccionado un examen habilitado en el sistema. La interfaz para el usuario consiste en una serie de preguntas de selección múltiple, las cuales deberá responder seleccionando la casilla de verificación para cada respuesta que se considere correcta. Para finalizar el examen se presiona el botón que activará el mensaje para validar las respuestas y asignar la nota obtenida.

Realización de examen en la aplicación Certinet

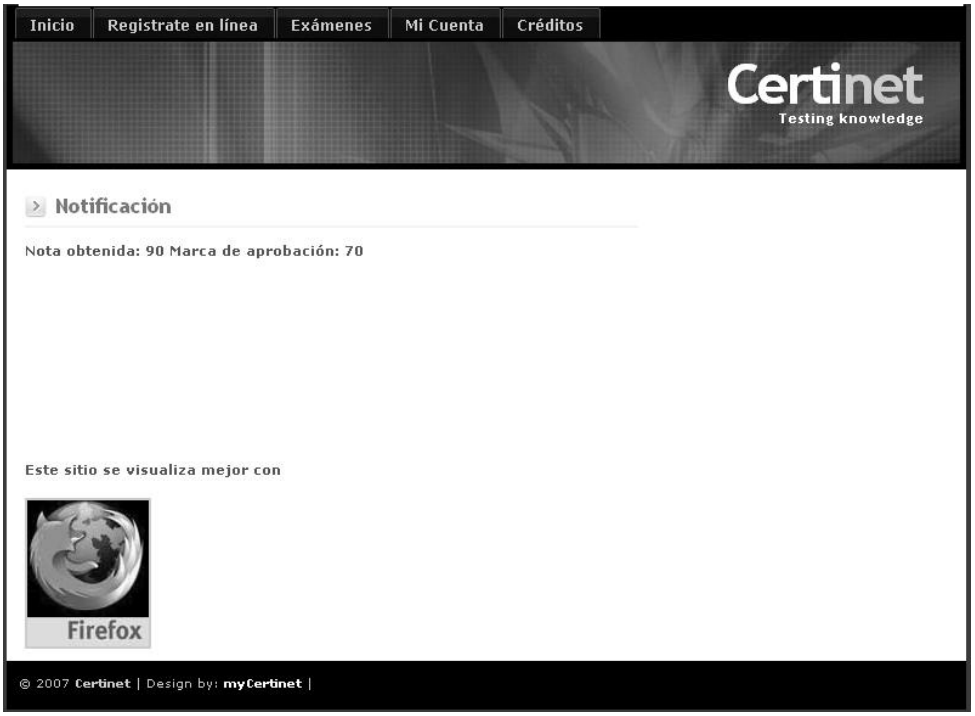
c. VALIDACIÓN DE LOS RESULTADOS OBTENIDOS. Al terminar de responder el examen, el servlet controlador recibe el mensaje "finalizar", a lo que responde enviando el mensaje que ejecutará los métodos respectivos en el bean de realización de servicios del negocio, el cual retornará un mensaje con la calificación obtenida.

La calificación obtenida se compara con el valor esperado y si los resultados son satisfactorios en los casos de prueba ejecutados, se concluye que el componente funciona correctamente. En caso contrario, se deberá identificar el problema usando herramientas como el depurador proveído por la herramienta para hacer las correcciones que sean necesarias.

Es importante señalar que si un error se presenta después de haber realizado varias corridas satisfactorias de la prueba, los resultados obtenidos se invalidan y después de haberlo corregido, se deberán ejecutar todos los casos de prueba nuevamente.

La siguiente figura muestra la información de salida obtenida al ejecutar el sistema con el caso de prueba del inciso anterior.

Notificación de resultado de examen en la aplicación Certinet



VII. RESULTADOS Y DISCUSIÓN

El modelo de trabajo propuesto y llevado a la práctica en los capítulos anteriores, es el resultado de la reunión de diferentes disciplinas y marcos de trabajo relacionados con la ingeniería de software y el análisis de sistemas orientados a objetos, que se combinan con el empleo de una tecnología en particular y con algunos enfoques y buenas prácticas usadas en el desarrollo profesional de aplicaciones para crear sistemas que ofrezcan flexibilidad para su mantenimiento y escalabilidad para soportar el crecimiento de la carga de trabajo a largo plazo.

El modelo se sustenta en tres pilares básicos: metodología orientada a objetos, arquitectura distribuida y componentes de software de código abierto, los que fueron escogidos con la finalidad de alcanzar los objetivos propuestos al inicio de la investigación, por lo que en esta sección se hace un análisis de los resultados obtenidos de su aplicación, en función de dichos objetivos.

A. DISEÑO Y MODELADO POR MEDIO DE UN SUBCONJUNTO DEL LENGUAJE UML

El lenguaje UML cuenta con trece tipos de diagramas diferentes pero no es una metodología para el diseño y modelado por sí sola, sino que su función se limita a proveer los recursos que pueden ser usados de acuerdo al criterio del diseñador del sistema. Los diagramas empleados en el modelo propuesto fueron suficientes para modelar las entidades y el comportamiento del sistema de acuerdo al enfoque planteado para la estructura y funcionamiento de cada uno de sus componentes, permitiendo hacer un diseño breve y conciso sin omitir aspectos relevantes del mismo.

B. BENEFICIOS OBTENIDOS DE LA ORIENTACIÓN A OBJETOS

De acuerdo a la literatura que trata el tema de la orientación a objetos de forma completa, los beneficios que ofrece son múltiples y de diferente índole, sin embargo, de

la aplicación hecha en la presente investigación, se pueden resaltar los siguientes aspectos:

- Facilidad de distribución del sistema en múltiples capas de funcionalidad débilmente acopladas, donde las capas superiores hacen uso de las capas inferiores, sin que las segundas tengan conocimiento de la existencia de las primeras.

- Seguridad en el manejo de la información de las entidades del modelo del negocio al acceder a sus atributos solamente a través de los métodos creados para tal propósito.

- Facilidad de manejo de la información de las entidades compuestas por múltiples objetos, esto como producto del encapsulamiento de toda la información en un solo objeto, que es recuperada en una sola operación de búsqueda, lo que permite eficiencia en la cantidad de líneas de código fuente escritas y reduce la complejidad al evitar la ejecución de múltiples consultas para recuperar toda la información relacionada con una entidad, lo que funciona de manera similar al almacenar la información del objeto en una base de datos.

- Reutilización del código fuente al crear paquetes de clases independientes que implementan funcionalidades que pueden ser usadas en otros módulos y/o proyectos, evitando la necesidad de volverlas a crear.

C. ARQUITECTURA DE IMPLEMENTACIÓN CON COMPONENTES DE SOFTWARE DE CÓDIGO ABIERTO

Derivado de la investigación realizada para conocer los beneficios más comunes de las aplicaciones de código abierto, destaca el hecho de que no giran alrededor de los ahorros económicos en los costos de adquisición del software, sino que en la calidad del

mismo, lo que se debe entre otras razones, a que se logra reunir una gran cantidad de talento que ni siquiera las empresas de desarrollo de software más grandes del mundo lograrían, aparte de que los cambios introducidos a las aplicaciones son más lentos y graduales debido a que son para satisfacer requerimientos de mejora y no para tomar ventajas competitivas provocadas por estrategias mercadológicas, lo que tiene un impacto positivo en la calidad de los productos.

Como consecuencia de lo anterior, el desarrollo del caso de aplicación fue exitoso y se logró alcanzar la funcionalidad deseada con la arquitectura de componentes propuesta al inicio, con la única diferencia de que el servidor de aplicaciones JBoss se sustituyó por Sun Java System Application Server.

La razón de dicho cambio no fue por algún problema que se haya presentado, sino que se debió a que la versión que soporta la especificación EJB 3.0 aún no ha sido liberada definitivamente y durante el tiempo que duró la investigación se encontraba en fase de prueba, lo que fue considerado como un aspecto que podría afectar el desarrollo de la misma.

Un aspecto de mucha relevancia para el proyecto es la amplia variedad de herramientas y marcos de referencia ampliamente usados por la comunidad de desarrollo de software que cuentan con servicios de consultoría y soporte que aseguran el éxito de los proyectos y un buen retorno de la inversión, siendo Java y la Web, una de las plataformas más comunes para el desarrollo de aplicaciones empresariales.

D. APLICACIÓN DEL MODELO VISTA CONTROLADOR EN EL MODELO DE TRABAJO PROPUESTO

La plataforma empresarial de Java está dirigida al desarrollo de aplicaciones bajo una arquitectura distribuida, sin embargo, es fácil perder el enfoque de la distribución en subsistemas cohesivos y débilmente acoplados, lo que trae como consecuencia que se

creen dependencias entre las capas de la aplicación, mezclándose funcionalidades de presentación, control y lógica del negocio en el mismo subsistema, perdiéndose la flexibilidad y la capacidad de escalabilidad, a la vez que se dificulta la legibilidad y el mantenimiento de los programas.

Durante el desarrollo del caso de aplicación se adoptó el enfoque modelo vista controlador desde la fase de diseño del sistema, lo que permitió tener un código fuente bien estructurado y con un patrón de diseño muy similar en todos los casos, en el que se puede decir que un servlet se utiliza únicamente para direccionar las peticiones que se reciben vía el protocolo http hacia los componentes que deberán procesarlas, luego, recibe el resultado y lo direcciona hacia una interfaz de presentación vía el mismo protocolo; un bean de sesión se usa para la realización de operaciones, cálculos e interacción entre objetos, lo que es aprovechado para implementar los servicios del negocio, y por último, una clase de entidad se usa para encapsular la información y el comportamiento de las entidades del dominio del negocio y sus relaciones con otras similares.

Manteniendo los principios de diseño explicados con anterioridad, se pueden crear aplicaciones con la flexibilidad necesaria para ser separadas y distribuidas por capas de funcionalidad en diferentes dispositivos de hardware y software para hacerlas fácilmente escalables. Por ejemplo, los servlets se podrían implementar en un servidor dedicado a la capa de servicios Web, los bean de sesión y las entidades, en un servidor de lógica del negocio, y la capa de almacenamiento persistente, en un servidor de base de datos.

VIII. CONCLUSIONES

1. Las aplicaciones de software de código abierto son una alternativa interesante para el desarrollo de aplicaciones sobre plataformas empresariales que pueden generar un buen retorno de la inversión sobre su uso en una empresa y también un recurso valioso para el desarrollo tecnológico de nuestro país, al poner a la disposición de personas emprendedoras, todo lo necesario para iniciar proyectos sostenibles a largo plazo, con la posibilidad de beneficiar a todo tipo de instituciones, haciendo uso de las tecnologías más ampliamente usadas como lo son el lenguaje Java Empresarial y la orientación a objetos.

2. El enfoque presentado por Robert S. Pressmann en su libro *Ingeniería de Software, un enfoque práctico*, en el que afirma que el primer paso para que una organización desarrolle software orientado a objetos, consiste en desarrollar las clases del dominio del negocio (Pressmann:2002:364), es aplicable en la práctica, ya que dicha librería de clases es el cimiento de todas las aplicaciones a desarrollar en el futuro.

3. La especificación de las interfaces de las clases, consistente en las condiciones que se deben cumplir antes y después de hacer uso de un servicio prestado por ellas, combinado con el uso de un marco de trabajo como JUnit, facilitan las pruebas modulares y disminuyen considerablemente la posibilidad de que las clases contengan errores después de haber sido integradas a una aplicación, debido a que en la especificación se indican las condiciones en las que una clase mostrará el comportamiento esperado y las pruebas de unidad permiten generar el código para comprobarlo en la etapa de desarrollo.

4. La brecha que existe entre los sistemas de bases de datos relacionales y las aplicaciones orientadas a objetos se ha venido cerrando cada vez más y en la actualidad se cuenta con implementaciones para el manejo de entidades persistentes con mapeo objeto-relacional de forma casi transparente para los desarrolladores, lo

que reduce considerablemente la complejidad que dicha tarea representaba en el pasado, incrementando la productividad del proceso de desarrollo de software y definiendo un poco más la tendencia sobre el uso de los sistemas de bases de datos relacionales para almacenar información proveniente de aplicaciones orientadas a objetos.

5. Al igual que como sucede con las aplicaciones propietarias en lo que a calidad y servicios de soporte se refiere, existen diferentes categorías para las aplicaciones de código abierto, en las que existen empresas a nivel mundial que desarrollan el software siguiendo procesos maduros y que cuentan con una infraestructura de soporte de diferentes niveles para sus plataformas empresariales.

6. Los recursos de donde se pueden obtener componentes de código abierto incrementan la productividad de los desarrolladores, debido a que no es necesario desarrollar un componente que se puede obtener a partir de uno existente, pudiendo modificarse el código fuente para adaptarlo a las necesidades y estándares de la organización, sin ninguna restricción.

7. Los recursos de software, documentación y marcos de referencia que se pueden obtener libremente y sin restricciones están al alcance de cualquier emprendedor de tecnología que quiera emplearlos para desarrollar aplicaciones empresariales sobre plataformas cuyos niveles de confiabilidad están comprobados.

8. Open source no significa “uso de software gratis sin ningún costo de propiedad”, sino que por el contrario, significa que se obtiene libremente y con acceso al código fuente, pero su uso involucra costos que hay que asumir, como servicios de soporte, consultoría y en una organización, contar con personal calificado para poder usarlo.

IX. RECOMENDACIONES

1. El mundo de las aplicaciones empresariales utilizando la plataforma Java y herramientas de código abierto es sumamente amplio y en ocasiones complejo, por lo que se recomienda apegarse a marcos de referencia existentes, que han sido creados para implementar soluciones más elaboradas que ayuden a incrementar la productividad eliminando la complejidad que puede surgir en aplicaciones de mayor tamaño, para aspectos como el enfoque modelo vista controlador, manejo de persistencia o desarrollo de aplicaciones Web, entre los que se pueden nombrar Struts, Hibernate, TopLink, Google WebToolkit, etc. La elección del más adecuado dependiendo de las necesidades, queda a criterio del evaluador.

2. Debido a que el alcance de la presente propuesta no abarca todos los aspectos relacionados con las diferentes capas de la arquitectura y los componentes de implementación de los subsistemas en que se divide una aplicación, es altamente recomendable capacitarse sobre aspectos vitales para el desarrollo de una aplicación empresarial como el funcionamiento y las capacidades de la API de persistencia, los recursos de programación existentes en el lenguaje Java y los criterios de elección de estructuras con funcionamientos similares para cada caso en particular, así como también los aspectos importantes de la funcionalidad del servidor de aplicaciones.

3. El desarrollo de software profesional requiere administrar adecuadamente las versiones de todos los archivos que componen un proyecto. Por tal razón, se recomienda el uso de un marco de trabajo para llevar a cabo dicha tarea, para lo que se propone el uso del sistema de versiones concurrentes (CVS por sus siglas en inglés), que es una aplicación de código abierto cuyo uso está soportado por el entorno de desarrollo integrado NetBeans. Se puede obtener mayor información sobre esta herramienta en el sitio de Internet http://ximbiot.com/cvs/cvshome/.

4. En un entorno empresarial, el rendimiento y desempeño de una aplicación es vital para satisfacer los requerimientos para los que ha sido creada, por tal razón, a pesar

de que el sistema se pueda implementar en componentes no empresariales como un sistema operativo para ejecutarse en un computador personal o un sistema de base de datos de edición estándar, no se recomienda hacerlo ya que no se estarían obteniendo los beneficios de escalabilidad y alta disponibilidad de la aplicación.

5. En la mayoría de los casos, las aplicaciones de código abierto se van desarrollando por medio de contribuciones hechas por diferentes personas e instituciones, y entre el lanzamiento de una versión y la subsiguiente, existen actualizaciones en las que se van incorporando nuevas funcionalidades y correcciones a errores. Algunas herramientas cuentan con opciones de configuración en las que se puede activar la opción de descarga y actualización automática, que por lo general es el comportamiento predeterminado de la herramienta, lo que puede ocasionar problemas en ambientes que se encuentran en producción al descargarse actualizaciones que ocasionen comportamientos diferentes a los que se tienen en uso, por lo que se recomienda hacer uso de dicha funcionalidad solamente para ambientes de prueba.

X. BIBLIOGRAFÍA

1. Bell, Douglas; Parr, Mike. 2003. *Java para estudiantes*. México. Prentice Hall

2. Bruegge, Bernd; Dutoit, Allen H. 1999. *Object-oriented software engineering, conquering complex and changing systems.* Carnegie Mellon University, School of Computer Science. Pittsburgh PA, USA. Prentice Hall.

3. Eriksson, Hans-Erik; Penker, Magnus, Lyons, Brian; Fado, David. 2004. *UML 2 toolkit*. Indianapolis, Indiana. USA. Wiley Publising, Inc.

4. Keith, Mike; Schincarlol, Merrick. 2006. *Pro EJB 3: Java persistence API.* USA. Apress.

5. Mukhar, Kevin; Zelenak, Chris; Weaver, James L.; Crume, Jim. 2006. *Beginning Java EE 5: from novice to professional.* USA. Apress.

6. Myatt, Adam. 2007. *Pro NetBeans IDE 5.5 enterprise edition.* USA. Apress.

7. O'Docherty, Mike. 2005. *Object-oriented analysis and design, understanding system development with UML 2.0. England*. John Wiley & Sons, Ltd.

8. Pentaho Inc. 2005. *Pentaho open source business intelligence platform technical white paper*. USA.

9. Pressmann, Roger S. 2002. *Ingeniería de software, un enfoque práctico.* Quinta edición. España. McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S.A.

10. Schwalbe, Kathy. 2006. *Information technology project management*. Fourth Edition. Canada. Thomsom Course Technology.

11. Sintes, Tony. 2002. *Teach yourself object oriented programming in 21 days*. USA. Sams Publishing.

12. Stojanovic, Zoran. 2005. *A method for component-based and service-oriented systems engineering, doctoral dissertation*, Delft University of Technology. The Netherlands.

13. Woods, Dan; Guliani, Gautam. 2007. *Open source for the enterprise: managing risks, reaping rewards*. USA. O'Reilly Media, Inc.