



**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**Facultad de Ingeniería en Electricidad y Computación**

**TÓPICO DE GRADUACIÓN**

“Módulo StoreFront de Eguana”

Previa a la obtención del Título de:

**INGENIERO EN COMPUTACIÓN ESPECIALIZACIÓN**

**SISTEMAS TECNOLÓGICOS**

Presentada por:

Erika Anabella Aizaga Barzola  
Federico Xavier Domínguez Bonini  
Wendy Carolina Ramos Dávila

GUAYAQUIL – ECUADOR

2005

## **A G R A D E C I M I E N T O**

*A Dios.*

*A nuestros padres y hermanos por ser  
nuestra compañía fiel en todo momento.*

*A nuestros compañeros de tópico.*

*A nuestro director de tópico, Ing. Luís  
Muñoz por todo su conocimiento*

*impartido.*

## DEDICATORIA

*A Dios por haberme iluminado y orientado en todo el proceso de desarrollo de esta tesis.*

*A mis padres Nelly y Luis por estar siempre apoyándome, motivándome para culminar este proyecto.*

*A mi hermana Patty por sus sabios  
consejos y palabras de aliento en todo  
momento.*

*Erika Anabella Aizaga Barzola*

## DEDICATORIA

*A mis padres por su apoyo incondicional.*

*Federico Xavier Domínguez Bonini*

# DEDICATORIA

*A Dios y a mi familia.*

*Wendy Carolina Ramos Dávila*



## TRIBUNAL DE GRADUACIÓN

---

Ing. Miguel Yapur

SUB-DECANO DE LA FIEC

---

Ing. Luis Muñoz

DIRECTOR DE TÓPICO

---

Ing. Gato Valverde

MIEMBRO PRINCIPAL

---

Ing. Cristina Abad

MIEMBRO PRINCIPAL

## **DECLARACIÓN EXPRESA**

“La responsabilidad del contenido de este Proyecto, nos corresponde exclusivamente; y el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral”

(Reglamento de exámenes y títulos profesionales de la ESPOL)

---

Erika Aizaga Barzola

---

Federico Domínguez Bonini

---

Wendy Ramos Dávila

## RESUMEN

Eguana es un proyecto inspirado en el concepto de E-Procurement. Entre muchas cosas, E-Procurement busca una convergencia entre proveedores y compradores en una empresa. Eguana implementa esta convergencia usando J2EE, demostrando el poder y alcance de esta tecnología. StoreFront es el módulo de Eguana encargado de manejar la interacción directa entre compradores y vendedores. En este módulo se usa el espectro completo de la tecnología J2EE desarrollado con diversas herramientas de código abierto, sirviendo así como una prueba de concepto de esta tecnología.

## ÍNDICE GENERAL

<a href="#"><u>CAPÍTULO 2.....</u></a>	<a href="#"><u>38</u></a>
<a href="#"><u>CAPÍTULO 3.....</u></a>	<a href="#"><u>132</u></a>
<a href="#"><u>CONCLUSIONES.....</u></a>	<a href="#"><u>284</u></a>
<a href="#"><u>Recomendaciones.....</u></a>	<a href="#"><u>290</u></a>
<a href="#"><u>Glosario de términos.....</u></a>	<a href="#"><u>292</u></a>
<a href="#"><u>ANEXO 1.....</u></a>	<a href="#"><u>305</u></a>
<a href="#"><u>Bibliografía.....</u></a>	<a href="#"><u>309</u></a>

## ÍNDICE DE FIGURAS

<b>Figura 1: Esquema de arquitectura Eguana.....</b>	<b>48</b>
<b>Figura 2: Esquema de aplicaciones multicapas [3].....</b>	<b>57</b>

Figura 3: Servidor J2EE y sus contenedores [3].....	65
Figura 4: Diagrama del patrón de diseño MVC [6].....	71
Figura 5: Funcionamiento de los Enterprise Beans [7].....	94
Figura 6: Diagrama de capas de JBOSS [10].....	107
Figura 7: Pantalla de Eclipse.....	117
Figura 8: Diagrama MVC (Modelo Vista Controlador) [15].....	123
Figura 9: Diagrama arquitectura Eguana con tecnología J2EE basada .....	138
Figura 10: Casos de uso.....	170
Figura 11: Diagrama de estado del objeto Compra.....	172
Figura 12: Diagrama estático de objetos.....	178
Figura 13: Escenario 9.1.....	179
Figura 14: Escenario 9.4.....	180
Figura 15: Escenario 10.1.....	181
Figura 16: Escenario 10.3.....	182
Figura 17: Diagrama estático de OrdenCompraSessionBean.....	183
Figura 18: Escenario 11.1.....	184

<b>Figura 19: Escenario 12.1.....</b>	<b>185</b>
<b>Figura 20: Escenario 13.1.....</b>	<b>186</b>
<b>Figura 21: Escenario 14.1.....</b>	<b>187</b>
<b>Figura 22: Autenticación basada en un formulario [3].....</b>	<b>192</b>
<b>Figura 23: Alerta previa a página de inicio de Eguana.....</b>	<b>198</b>
<b>Figura 24: Página de inicio de Eguana.....</b>	<b>200</b>
<b>Figura 25: Página de menú de Eguana.....</b>	<b>203</b>
<b>Figura 26: Funcionamiento del catálogo.....</b>	<b>228</b>
<b>Figura 27: Funcionamiento del carrito de compras .....</b>	<b>235</b>
<b>Figura 28: Diagrama del proceso de compra.....</b>	<b>242</b>
<b>Figura 29: Estructura del patrón MVC.....</b>	<b>258</b>
<b>Figura 30: Diagrama entidad relación.....</b>	<b>267</b>
<b>Figura 31: Pantalla de inicio de sesión.....</b>	<b>270</b>
<b>Figura 32: Sección de compras con el catálogo.....</b>	<b>272</b>
<b>Figura 33: Catálogo de productos.....</b>	<b>272</b>
<b>Figura 34: Carrito de compras.....</b>	<b>274</b>
<b>Figura 35: Confirmación del carrito de compras.....</b>	<b>275</b>

**Figura 36: Consulta de órdenes.....276**

**Figura 37: Pantalla inicio de Eguana.....277**

## **ÍNDICE DE TABLAS**

**Tabla 1: Funcionalidad de StoreFront y APIs de J2EE utilizados.....67**

**Tabla 2 : Estructura del patrón de diseño MVC [5].....70**

**Tabla 3: Sumario de tipos de Enterprise Beans [3]...89**

**Tabla 4: Comparación JBOSS y competencia [16].....129**

**Tabla 5: Análisis horas - hombres.....134**

**Tabla 6: Análisis de costo.....136**

**Tabla 7: Roles vs. opción de menú.....201**

**Tabla 8: Atributos del formulario de login.....209**

**Tabla 9: Clases del catálogo.....226**

**Tabla 10: Páginas del catálogo.....227**



<b>Tabla 11: Clases del modelo del carro de compra.....</b>	<b>232</b>
<b>Tabla 12: Páginas de la vista del carro de compra.....</b>	<b>233</b>
<b>Tabla 13: Clases controladoras del carro de compra.....</b>	<b>234</b>
<b>Tabla 14: Objetos del proceso de compra.....</b>	<b>240</b>
<b>Tabla 15: Clases para mensajería en Storefront.....</b>	<b>247</b>
<b>Tabla 16: Parámetros en descriptores XML para mensajería.....</b>	<b>248</b>
<b>Tabla 17: Estados de las órdenes de compra.....</b>	<b>254</b>
<b>Tabla 18: Objetos del proceso de despacho.....</b>	<b>256</b>
<b>Tabla 19: Recursos compartidos en Eguana.....</b>	<b>279</b>
<b>Tabla 20: Integración vía enlaces.....</b>	<b>283</b>
<b>Tabla 21: Estructuras de base de datos.....</b>	<b>307</b>
<b>Tabla 22: Objetos J2EE de la aplicación bancaria.....</b>	<b>308</b>

# INTRODUCCIÓN

El presente trabajo es una descripción del proyecto Eguana y específicamente de su módulo StoreFront. Eguana fue desarrollado completamente con herramientas y plataformas de código abierto. El objetivo de este trabajo es demostrar la capacidad completa de la plataforma J2EE y del movimiento código abierto. Hasta la escritura de este documento Eguana es el único proyecto código abierto en su clase.

La organización de este documento se detalla a continuación:

El primer capítulo "Antecedentes" explica todos los conceptos y definiciones que conforman un E-Procurement. Además la descripción, objetivos, alcance y visión de Eguana.

El segundo capítulo “El sistema Eguana” presenta los conceptos y las tecnologías consideradas para el desarrollo del sistema y las herramientas de código abierto que se utilizaron para su implementación.

El tercer capítulo “Implementación del módulo StoreFront de Eguana” describe el diseño, análisis e implementación del módulo StoreFront como una solución Web que utiliza las especificaciones y componentes de la tecnología J2EE.

Por último, se presentan las conclusiones y recomendaciones, bibliografía, glosario y anexos.

# **CAPÍTULO 1**

## ***1. Antecedentes***

### **1.1 E-Procurement**

E-Procurement, en términos sencillos “solicitud o gestión electrónica”, se define como la automatización de los procesos de solicitudes de una organización utilizando aplicaciones basadas en el Web.

Así como los sistemas ERP<sup>1</sup> desarrollan tareas para automatizar sus procesos internos, E-procurement reúne a compradores y proveedores

---

<sup>1</sup> ERP: Enterprise Resource Planning

geográficamente dispersos para interactuar y ejecutar transacciones de compra sobre el Internet.

En un sistema E-procurement basado en el Web, cada paso del proceso de solicitud ocurre electrónicamente; por tanto, estas aplicaciones bajan los costos de procesos e inventarios, extienden el alcance de los proveedores y mejoran el acceso de los mismos por parte de los clientes.

### **1.1.1 Componentes**

Un sistema E-procurement, entendido como una realización de compras a través del Internet, tiene un nivel de eficacia y eficiencia que depende de una serie de sistemas relacionados: [1]

E-Fulfillment, que se define como cualquier actividad que se realiza en el comercio electrónico desde que se acepta

el pedido por un proveedor hasta que el producto es recibido y aceptado por el comprador o devuelto.

E-marketplace, que es un punto de encuentro, diseñado como una página Web, donde se ponen en contacto empresas compradoras y vendedoras para llevar a cabo sus relaciones de compra-venta. Se puede decir que son comunidades de negocios donde los participantes comparten información que mejora la eficiencia de toda la cadena y donde pueden participar compradores, fabricantes, vendedores, transportistas, bancos que financian la operación y hasta la empresa de seguros que la avala.

E-payment, definido como la realización electrónica del pago en la cuenta del banco del proveedor.

Teniendo en cuenta lo descrito, podemos decir que implementar un sistema e-procurement en una empresa supone automatizar y agilizar los procesos de compra así como automatizar y agilizar los pagos (e-payment) a proveedores, a través de un e-marketplace y apoyados en un sistema e-fulfillment.

### **1.1.2Tipos**

Debido a la diferenciación previa del producto comprado, se presentan tres tipos de un sistema e-procurement: [1]

E-procurement simple: Referido a la compra de bienes y servicios que requieren un proceso sencillo de compra.



E-procurement complejo: los bienes y servicios de adquisición son de una relevancia tal que se requiere una selección y evaluación previa de los posibles proveedores.

E-procurement estratégico: Se ejercita el sistema con un pequeño grupo de proveedores preseleccionados y evaluados capaces de garantizar un buen precio y poder reaccionar ante una demanda imprevista.

Otras tipologías e-procurement pueden ser: [1]

E-transacting: Compras de cantidades pequeñas (día a día) de bienes no inventariables.

E-sourcing: Compras de volumen casi siempre centralizadas para toda un área definida (con subasta invertida). También puede ser una subasta normal en la que se puede ofrecer bienes inventariados en desuso.

E-intelligence: Compra negociada y programada que es posible en la medida en que se ahorre tiempo y esfuerzo en las actividades previamente señaladas.

### **Elementos de un sistema E-procurement**

En la puesta en práctica de un sistema e-procurement se pueden diferenciar los siguientes elementos: [1]

Un registro de proveedores.

Un catálogo de productos.

Un soporte de proceso o programa que permite la compra electrónica.

Una subasta inversa, mediante la cual la empresa compradora no realiza los pasos tradicionales de búsqueda de proveedores sino que los proveedores se acercan a la empresa compradora y “pujan” por conseguir la adjudicación de la orden de compra.

### **1.1.3 Etapas de un proceso de compras**

Una forma de reducir los costos en una compra es actuando sobre las diferentes actividades que se pueden identificar en un proceso de compras:

Identificación de la necesidad percibida para un ítem por parte del comprador.

El ítem es escogido del stock ofrecido por socios de negocios o de un catálogo “en línea”. La elección automáticamente genera una solicitud formal para la autorización.

La autorización es encaminada a una orden de compra generada y despachada a través de canales de comunicación (Internet, VPN<sup>1</sup>).

Recepción de la solicitud por el vendedor.

Despacho del ítem por el vendedor a través de canales de distribución normales.

Recepción y chequeo del ítem por el comprador y pago automático del bien recibido.

## 1.2 ¿Qué es Eguana?

Eguana es un sistema de información comercial orientado a la administración y soporte del proceso de compra y venta entre empresas en el Web.

---

<sup>1</sup> VPN: **Virtual Private Network**: es un túnel de encriptación entre dos redes privadas por intermedio de una red pública (por ejemplo: Internet)

Basado fundamentalmente en el concepto de los sistemas E-Procurement, Eguana brinda a compradores y vendedores un sitio de encuentro en Internet donde pueden realizar solicitudes de productos de manera electrónica, tanto a través de un proceso de compra simple como de una subasta o licitación, donde el sistema proporcionará los principios propios de estos dos tipos de actividades comerciales de una manera ágil y eficiente.

El sistema Eguana para cumplir con las expectativas del mercado competitivo actual y proveer todas las herramientas necesarias para un rápido y completo comercio electrónico, posee cuatro módulos que separan las diferentes funciones de un portal de compras:

Administración

StoreFront

Subasta y Licitación

Reportes gerenciales

### 1.3 Objetivos y alcance de Eguana

#### Objetivos:

Desarrollar un mercado en línea en el que las empresas compradoras y los proveedores pueden compartir información y efectuar transacciones.

Proporcionar información general y comercial de los productos registrados en el catálogo a tantos usuarios como sea posible.

Ofrecer métodos de búsqueda de productos y características de los mismos de manera sencilla, contando con una sólida infraestructura de gestión del contenido, la cual permite disponer de un catálogo en línea de alta calidad.

Mejorar los tiempos de espera que se producen en los procesos de gestión de compras.

Monitorear la gestión de compra que se ha realizado.

Brindar varios canales de comunicación (email, mensajería) entre los usuarios del sistema dando un valor agregado para poder realizar eficientemente las negociaciones entre empresas participantes.

Producir una reducción en los costos de adquisición de los productos.

Proporcionar una herramienta Web que permita generar reportes gerenciales de las transacciones de ventas y/o compras como soporte a la toma de decisiones, basándose en información integrada y global del negocio.

Proporcionar el servicio de licitaciones y subastas.

Aprovechar los conceptos de E-Procurement, soluciones que contribuyen a hacer más eficiente el proceso total de compras, reduciendo costos de transacción y de operación, aumentando el control de procesos y gastos a través de una aplicación Web que resuelva las barreras existentes y aporte valor a los medios de compras ya establecidos en su empresa.

### **Alcance**

Desarrollar una aplicación de manera integrada que brinde un valor diferenciado combinando los sistemas y los procesos de negocios con el alcance de la tecnología Internet no es una actividad fácil, se debe evaluar alternativas tales como: E-Business y E-Procurement.

Teniendo en cuenta lo descrito y lo definido en la sección 1.1 un sistema E-Procurement es la automatización de los procesos de solicitudes de una organización. Existen varios tipos de E-Procurement para implementar, por lo cual es necesario decidir cual es el más idóneo y adaptable a las necesidades del sistema.



Eguana consideró en su análisis las bases del concepto de un E-Procurement complejo, puesto que se basa en la selección y evaluación de los posibles proveedores, otorgando fiabilidad a los clientes.

Analizando el mercado actual empresarial, las compañías medianas y grandes, gastan hasta un 20% más por no planificar y negociar sus compras de insumos y servicios, dando lugar a que las compras a último momento, la ausencia de negociación de precios con los proveedores, la falta de recursos optimizados, entre otros, aumenten los costos de las adquisiciones de una empresa. Por estas razones se combinaron las definiciones de los sistemas E-marketplace y E-payment, puesto que por lo analizado en la sección 1.1, las aplicaciones basadas en un E-Procurement bajan los costos de procesos e inventarios antes mencionados, extienden el alcance de los proveedores y mejoran el acceso de los mismos por parte de los clientes.

Evaluando todos estos puntos Eguana es una solución cuyo alcance permite:

Gestionar a los usuarios.

Gestionar Catálogo electrónico.

Gestionar los Proceso de compra, subasta y licitación.

Circuito de aprobación (Simulación de Pagos en Banco).

Seguimiento de los procesos.

Informes para toma de decisiones.

## **1.4 Visión de Eguana**

Internet se convirtió en la plataforma que abre las puertas a una nueva generación de negocios.

Por eso Eguana cree que aportando con un sistema que agiliza la gestión de compra mediante Internet, contribuye su ingreso al mundo en línea, utilizando tecnología de vanguardia y abierta que puede ser ejecutada en cualquier sistema operativo y acoplarse a diferentes plataformas tecnológicas, aportando una visión de los casos de negocios viables en la actualidad, logrando que en lugar de un simple sitio Web su presencia en Internet se convierta en una verdadera fortaleza para su negocio.

### **Trabajos relacionados**

Existe un gran número de soluciones E-Procurement parecidas a Eguana en el mercado. A continuación una lista de las más populares e importantes:

Purchasing +Plus. [www.palmasdev.com](http://www.palmasdev.com)

Purchase and Pay. Departamento de Trabajo del Reino Unido. <http://europa.eu.int/idabc/en/document/1011/320>

Ketera eProcurement: Capture Savings. [www.ketera.com](http://www.ketera.com)

En el mercado existen muchos más productos de los arribas listados, muchos difícilmente califican como E-Procurement a pesar de indicar lo contrario. Algunas soluciones son hechas para industrias específicas, otras son de propósito general. La mayoría han sido implementadas como software propietario y con licencias de altos precios inalcanzables para empresas pequeñas.

Soluciones E-Procurement código abierto hay muy pocas. En Sourceforge.net existe un solo proyecto de esta característica el cuál no está terminado aún. En Freshmeat.net no existe ninguno. Purchase and Pay es código abierto, funciona sobre Linux, y apareció en encabezados de la prensa especializada por ser la primera y única solución E-Procurement código abierto [2].

Lamentablemente Purchase and Pay es una solución “hecha en casa” para el departamento de trabajo del Reino Unido y no esta disponible en el mercado.

Eguana es hasta ahora el único proyecto E-Procurement de propósito general código abierto, el cuál puede ser implementado por cualquier empresa a un bajo costo.

# CAPÍTULO 2

## ***2. El sistema Eguana***

### **2.1 Eguana como prueba de concepto J2EE**

Hoy en día, cada vez más desarrolladores quieren hacer aplicaciones transaccionales distribuidas para empresas con la velocidad, seguridad y confiabilidad de la tecnología “server-side”<sup>1</sup>. Como es conocido, en el mundo competitivo del comercio electrónico o “e-commerce” y en la tecnología de información, las aplicaciones

---

<sup>1</sup> Server-side: Tecnología del lado del servidor

empresariales han tenido que ser diseñadas, construidas y producidas a menor costo, a mayor velocidad y con pocos recursos, en comparación a tiempos pasados.

Para reducir costos y agilizar el proceso de diseño y desarrollo de aplicaciones, la tecnología J2EE<sup>2</sup> provee toda una especificación basada en componentes para el diseño, desarrollo, ensamblaje y despliegue de aplicaciones empresariales. Esta plataforma ofrece un modelo de aplicación distribuida multicapa, componentes reusables, un modelo de seguridad unificado, control flexible de transacciones y soporte de servicios Web a través de un intercambio de datos integrados en protocolos y estándares abiertos basados en XML<sup>3</sup> [3].

Esta nueva especificación está siendo ampliamente utilizada y empleada en el diseño de aplicaciones Web distribuidas y hasta en aplicaciones cliente-servidor.

---

<sup>2</sup> J2EE: Java 2 Enterprise Edition

<sup>3</sup> XML: **eXtensible Markup Language** Es en la actualidad el formato estándar de intercambio de datos.

Debido a su gran potencia y su extendido uso, se ha escogido esta tecnología para que sea el fundamento de la funcionalidad del sistema Eguana.

### **2.1.1 Módulos de Eguana**

El Sistema Eguana esta constituido de los siguientes módulos:

Módulo de Administración

Módulo de StoreFront

Módulo de Licitación y subastas

Módulo de Reportes gerenciales

#### **Módulo de Administración**

Garantiza la realización de todos los servicios necesarios para la gestión, actualización y mantenimiento de los registros de empresas y productos.



Las empresas para poder ser registradas en el sistema deben ingresar una solicitud, la cual debe ser analizada y aprobada por el usuario administrador del sistema.

Luego de la aprobación del registro de la empresa, los usuarios de la misma pueden ingresar los productos que se encuentran en stock.

Este módulo también controla la configuración de las variables globales del sistema.

### **Módulo de StoreFront**

El módulo de StoreFront permite al usuario seleccionar los productos a través de un catálogo en línea que ha sido alimentado por las empresas vendedoras a través del módulo de administración del sistema. Una vez seleccionado el o los productos son almacenados en un carrito de compras, concepto conocido para una transacción de compras en Internet, lo cual facilita al cliente la comprensión del proceso.

Ordenados todos los productos se hace efectiva una orden de compra electrónica que envía notificaciones a las empresas vendedoras de los productos seleccionados y da inicio al proceso de aprobación del despacho dependiendo del resultado del pago de la orden a través de una aplicación bancaria interna del sistema, que contiene cuentas de las empresas registradas, donde se procesan los correspondientes débitos o créditos para la transacción de compra específica.

El sistema mantiene en todo momento notificado a las empresas del estado de las órdenes, ya sea del lado del comprador o del vendedor, en el módulo de StoreFront, a través de notificaciones propias del sistema como por vía email.

El proceso de compras llega a término cuando se ha realizado con éxito el proceso de pago y se da autorización al despacho.

## **Módulo de Licitación y subasta**

Garantiza la realización de todos los servicios necesarios para la gestión de licitación y subasta.

Las funcionalidades de este módulo:

- **Licitación:**
  - Recolección de requerimientos por unidades.
  - Consolidación de los requerimientos.
  - Publicación de la licitación.
  - Oferta de los licitantes.
  - Elección de la licitación ganadora.
  - Generación de una orden de compra.
  
- **Subastas:**
  - Publicación de la subasta.
  - Oferta de los subastantes.
  - Elección de la subasta ganadora.

### **Módulo de Reportes gerenciales**

Este módulo se encarga de generar todo tipo de reportes necesarios para los usuarios de Eguana. Los reportes pueden ser generados de una manera dinámica usando la herramienta JasperReports. Cada usuario, de acuerdo a su rol en el sistema, tiene la capacidad de ver su historial de compras, ventas, subastas, licitaciones y cualquier otro tipo de transacciones que existan o aquellas que la empresa ejecute en el sistema. La forma, contenido, formato y alcance de los reportes es altamente configurable gracias a la flexibilidad que JasperReports provee.

### **Módulo Bancario**

Dentro del proceso de compras, el siguiente paso a la generación de la orden es el pago electrónico del monto al proveedor. Para realizar este proceso, se creó un módulo que simula un sistema bancario, que actúa como una institución financiera que mantiene toda la información referente a los pagos de valores monetarios involucrados en las

transacciones de compras. Este módulo trabaja con las cuentas de las empresas compradoras y vendedoras para realizar las transacciones de débito o crédito correspondientes.

El pago electrónico se realiza por orden generada. Esto se debe a que si los productos adquiridos por la empresa compradora pertenecen a n empresas vendedoras, se generan n órdenes de compra para poder efectuar el pago a cada empresa implicada.

Una vez generadas las órdenes, con estado “En Proceso”, el pago se inicia con la verificación, por parte del módulo Storefront, del saldo de la cuenta de la empresa compradora, a través de un mensaje emitido al módulo bancario. Realizada la verificación, el módulo bancario envía un mensaje al módulo Storefront y procede al cambio de estado de la orden de compra, dependiendo del resultado. Si existe saldo se realiza la transacción del pago, debitando de la cuenta de la empresa compradora y acreditando a la empresa vendedora el valor de la orden correspondiente, y ésta pasa al estado “Pagada”. De ser negativo el

resultado, la orden pasa a estado “No Aprobada” y se da por terminado el proceso de compra de dicha orden.

Esta comunicación se lleva a cabo gracias a la existencia de una aplicación de mensajería que incluye a los dos módulos, lo que se explicará posteriormente.

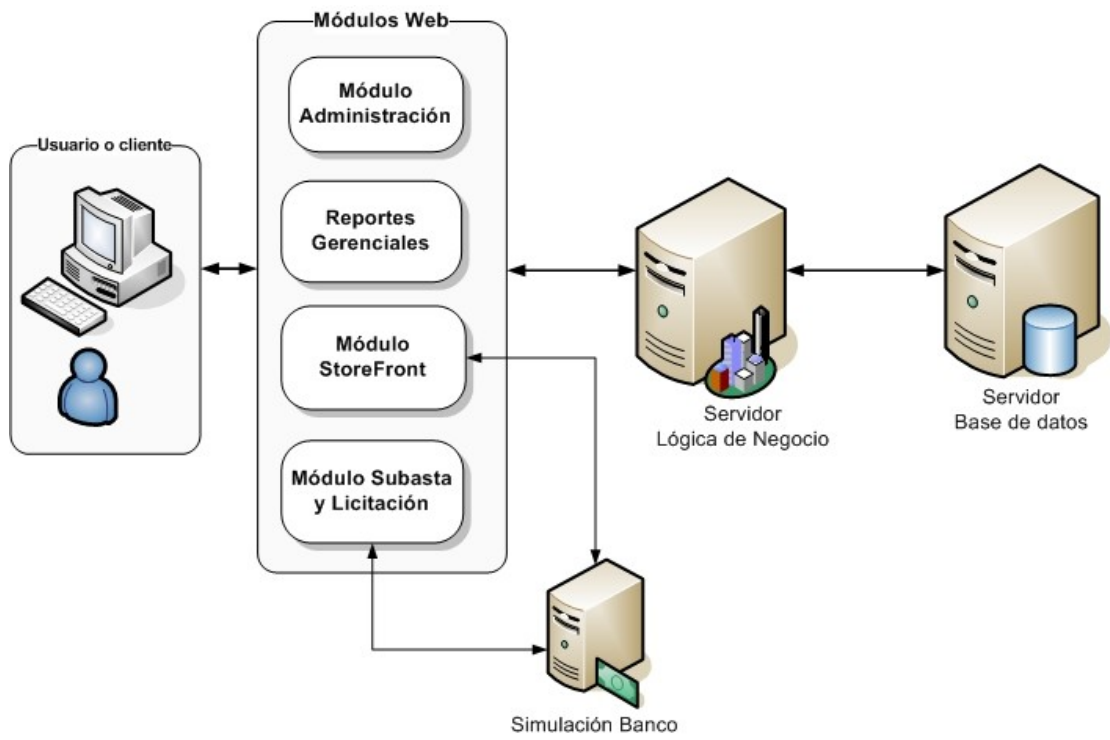
### **2.1.2 Arquitectura de Eguana**

Los sistemas que gestionan procesos de compras en Internet, se ajustan a un tipo de aplicación común: aplicaciones de comercio electrónico basados en Web. Por lo tanto, Eguana está implementada en una arquitectura empresarial estándar para este tipo de aplicaciones, es decir, una arquitectura multicapas con tecnología EJB<sup>1</sup> que es una propuesta de una arquitectura que propone separar la lógica de

---

<sup>1</sup> EJB: **Enterprise Java Beans**. Es un conjunto de componentes Java, desarrollados por Sun Microsystems, que permiten desarrollar aplicaciones distribuidas y multicapas.

presentación de la lógica de negocios basándose en componentes configurables y reutilizables, esto se explicará al detalle en la sección 3.1 que se refiere al análisis del módulo StoreFront.



**Figura 1: Esquema de arquitectura Eguana**



La aplicación tendrá instalado un servidor de acceso Web, que sirva páginas Web con componentes Java, que son los estándares del mercado en este tipo de operaciones. En cualquier caso, es imprescindible que el medio de acceso sea un navegador de páginas HTML<sup>1</sup>, tales como Internet Explorer o Netscape.

Los módulos Web que conforman el sistema (Módulo de Administración, Módulo StoreFront, Módulo Subasta y Licitación y Reportes Gerenciales), han sido diseñados para operar cada uno independiente del otro, pero con la capacidad de integrarse, para cubrir todo el ciclo de vida de la actividad comercial en un portal de compras.

Para realizar transacciones o consultas en cada uno de los módulos, estos deberán interactuar con un servidor que contiene la lógica de negocio, y un servidor de base de datos.

---

<sup>1</sup> HTML: **Hypertext Markup Language**. Lenguaje para escribir documentos en el web.

El servidor de lógica de negocio será el que contenga las reglas definidas para la ejecución de cada transacción y ejecute las transacciones. El servidor de base de datos será donde residen los datos de la empresa, deberá contar con una base de datos MySQL<sup>1</sup>, para ejecución de las transacciones. Los entornos de aplicaciones más modernos para estos ambientes son J2EE y .Net. Ambos servidores deberán ser de disponibilidad 100%.

El servidor de interfaz será el servidor o grupo de servidores requerido para interactuar con los usuarios; será el que les provea de las interfaces necesarias y vincule la transacción requerida con el servidor de lógica de negocios.

Los módulos StoreFront, Licitación y Subasta interactúan con la simulación de un sistema E-payment (módulo bancario) definido como la realización electrónica del pago en la cuenta del banco del proveedor.

---

<sup>1</sup> MySQL: Base de datos código abierto.

## 2.2 El módulo StoreFront y J2EE

El modulo "StoreFront" es la sección del sistema que se encarga de implementar la funcionalidad de "tienda", de ahí su nombre (Store = tienda, front = frente). La funcionalidad de tienda es básicamente:

1. Ingresar a la tienda (ingreso al sistema, login)
2. Buscar en las perchas el o los producto que deseo comprar (navegación en el catalogo)
3. Separar los productos que deseo comprar en una canasta o carrito de compras. (Carro de compras virtual).
4. Llevar los productos en el carrito a la cajera. (Orden de compra virtual).
5. Pagar los productos. (Transacción de pago).
6. La cajera me entrega el o los productos cancelados. (Orden de entrega virtual).

Además de implementar estos pasos análogos a una tienda o supermercado físico debemos implementar funcionalidad exclusiva de una tienda virtual Web la cuál es:

7. Automatización de comunicación entre comprador y vendedor vía email.
8. Comunicación al vendedor del estado de sus ventas vía email y mensajes del sistema.
9. Comunicación al comprador del estado de sus compras vía email y mensajes del sistema.
10. Comunicación al comprador y al vendedor de eventos de la transacción. Ejemplo: anulación de orden, negación de crédito, etc.

### **2.2.1 Demostración del uso de la tecnología J2EE.**

Para verificar la demostración del uso de la especificación J2EE en el módulo StoreFront de Eguana, es indispensable conocer acerca de esta especificación, sus orígenes y de sus principales fundamentos.

### **Origen y Definición de J2EE**

En los años 90's surge en el mundo de la tecnología de información la necesidad de las empresas de dar un giro real a sus sistemas de gestión que automatizan sus procesos, debido a la presencia latente del Internet y de la intranet como parte del ciclo de trabajo y a la gran demanda del comercio electrónico (e-commerce). Existían varios puntos de discusión que promovieron ciertos cambios: [4]

Necesidad de migrar los sistemas de información empresariales (EIS Enterprise Information Systems) de arquitecturas en dos capas (cliente/servidor) a arquitecturas más flexibles en tres o más capas.

Evolución de los servicios de middleware OTM, MOM,

ORB's.

Aumento en el uso de internet e intranets para aplicaciones empresariales.

Surge entonces la necesidad de definir un estándar para el desarrollo de aplicaciones basadas en arquitecturas de componentes en varias capas. Sun y sus socios industriales deciden generar el estándar que permitirá desarrollar aplicaciones de gran magnitud que sean livianas, eficientes y a bajo costo, basándose en las tecnologías existentes como JSP, Servlets, JDBC, EJB, etc.

Así nace la plataforma Java 2 Enterprise Edition (J2EE), que es una especificación para el diseño, desarrollo, ensamblaje y despliegue de aplicaciones empresariales y aplicaciones basadas en el Web. El objetivo principal de J2EE es crear un simple modelo de desarrollo para aplicaciones empresariales utilizando componentes basados en el

modelo de aplicación. Por tanto, la plataforma J2EE consiste de un conjunto de servicios, interfaces de programación de aplicaciones (APIs<sup>1</sup>) y protocolos que proveen la funcionalidad para desarrollar aplicaciones Web multicapas.

### **Características**

Entre sus características principales están: [4]

Define un mecanismo estándar basado en el concepto de portabilidad del lenguaje Java.

Pretende facilitar y disminuir el costo de desarrollo de EIS<sup>2</sup> de múltiples capas que permitan adaptarse a los requerimientos actuales.

Fomenta la separación en capas y el desarrollo especializado de componentes.

---

<sup>1</sup> API: De sus siglas en inglés Application Programmable Interface.

<sup>2</sup> EIS: De sus siglas en inglés Enterprise Information Server. Servidor de información empresarial.

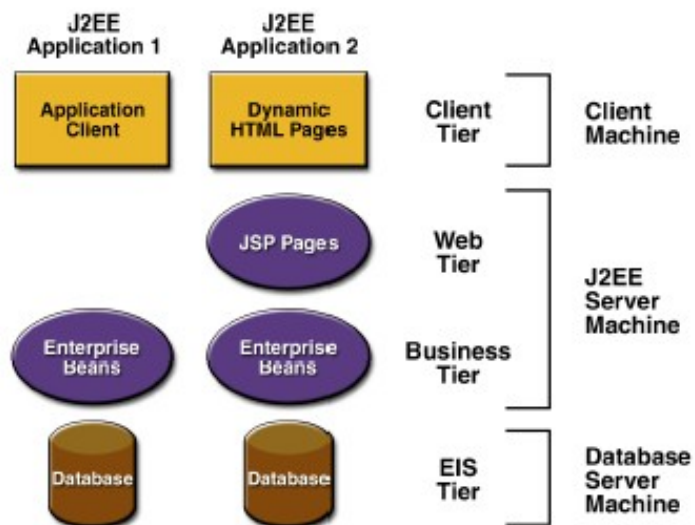
Disminuye el código necesario para desarrollar los sistemas a través de servicios ya provistos y APIs que implementan las funcionalidades normalmente requeridas por los EIS.

### **Arquitectura**

El modelo de aplicaciones J2EE se basa en una arquitectura multi-capa.

La lógica de la aplicación es dividida en componentes de acuerdo a su función y son instalados en diferentes máquinas dependiendo de la capa en el ambiente J2EE a la cual el componente pertenece.





**Figura 2: Esquema de aplicaciones multicapas [3]**

La figura 2 muestra dos aplicaciones J2EE multicapa divididas en las capas descritas a continuación: [3]

Componentes de la capa del cliente, que corre en la máquina cliente.

Componentes de la capa Web, que corre en el servidor J2EE.

Componentes de la capa de negocios que corre en el servidor J2EE.

Software de la capa EIS que corre en el servidor EIS.

Aunque una aplicación J2EE pueda consistir de las 3 o 4 capas mostradas en la gráfica, las aplicaciones multicapas son generalmente consideradas a ser de 3 capas debido a que son distribuidas en 3 diferentes localidades: máquina del cliente, la máquina del servidor J2EE y la máquina destinada a la base de datos.

### **Componentes**

Las aplicaciones J2EE están formadas por componentes. Un componente J2EE es una unidad de software funcional que es

ensamblado en una aplicación J2EE con sus clases y archivos relacionados y que se comunica con otros componentes. La especificación J2EE define los siguientes componentes: [3]

Aplicaciones clientes y applets<sup>1</sup> que son componentes que corren en el cliente.

Componentes de tecnología JSP (Java Server Pages) y Java Servlet que son componentes WEB que corren en el servidor.

Componentes EJB (Enterprise JavaBeans) que son componentes de negocios que corren también el servidor.

Los componentes J2EE son escritos en lenguaje Java y son compilados de la misma manera que cualquier programa en dicho lenguaje. La diferencia entre los componentes J2EE y clases de Java simples es que los componentes J2EE son ensamblados dentro de una aplicación J2EE

---

<sup>1</sup> Applets: Aplicaciones de java que se ejecutan dentro de un navegador Web.

y verificados que cumplan la especificación J2EE. Además se realiza con ellos un despliegue de la aplicación en producción, donde son ejecutados y manejados por el servidor J2EE.

### **APIs de J2EE**

J2EE también es catalogado como un ambiente servidor para aplicaciones distribuidas o multicapas que posee los siguientes elementos:

Una infraestructura para soportar las aplicaciones, conocida como J2EE runtime.

Un conjunto de Java APIs para construir las aplicaciones.

J2EE ha basado su estándar en algunos APIs de Java para armar las diferentes capas de una aplicación. Los más conocidos son<sup>1</sup>:

---

<sup>1</sup> Estas versiones son usados por J2EE 1.4, la última versión de J2EE actualmente en producción.

JSP 2.0: API que permite la generación dinámica de páginas Web combinando código HTML, WML o XML y elementos JSP.

Servlet 2.4: API que permite definir clases servlets HTTP específicas.

EJB 2.1: API que permite la creación de componentes que almacenan la lógica del negocio.

JNDI 1.2: API que permite el acceso a servicios de nombres o directorios.

JDBC 2.0 Extensión: API para el acceso a base de datos relacionales con soporte de pool de conexiones y transacciones distribuídas.

JTA 1.0: API de alto nivel que permite controlar transacciones distribuídas.

JMS 1.1: API estándar de mensajería que permite a los componentes de una aplicación J2EE crear, enviar, recibir y leer mensajes.

JavaMail 1.3: API estándar independiente del protocolo que permite el envío y recepción de mails.

RMI/IIOP 1.0: API para la invocación remota de métodos con soporte para protocolos IIOP <sup>1</sup> y JRMP<sup>2</sup>.

### **Contenedores J2EE**

Normalmente, las aplicaciones clientes son difíciles de codificar porque involucran muchas líneas de código para manejar transacciones, multihilos y otros detalles complejos de bajo nivel. La arquitectura J2EE, que es independiente de plataforma y basada en componentes, hace a las aplicaciones J2EE fáciles de escribir debido a que la lógica del negocio está organizada dentro de componentes reusables. Además, el servidor J2EE provee servicios en la forma de un contenedor para cada tipo de componente.

---

<sup>1</sup> IIOP: Es el protocolo inter-ORB de Internet. En particular, IIOP se usa como el protocolo de transporte para una versión de Java RMI.

<sup>2</sup> JRMP: Es el protocolo de Método Remoto de Java. Protocolo de comunicación entre objetos distribuidos que trabajan en la tecnología Java RMI.

Un contenedor es un programa que administra y da soporte de ejecución a los componentes de aplicación. Provee acceso a los APIs de J2EE para que sean usados por los componentes de aplicación. Permite añadir servicios de una manera transparente en beneficio de los componentes de aplicación (JDBC<sup>1</sup>, JNDI<sup>2</sup>, JTS<sup>3</sup>) y deben proveer un ambiente de ejecución compatible con J2SE.

En el Web, para que un componente EJB o cliente pueda ser ejecutado, deben ser ensamblados dentro una aplicación J2EE y realizar su despliegue dentro de su contenedor correspondiente.

### **Tipos de Contenedores**

J2EE especifica los siguientes tipos de contenedores: [3]

---

<sup>1</sup> JDBC: Es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

<sup>2</sup> JNDI: Es el sistema de nombrado en Java. JNDI define un espacio de nombres en forma de árbol en el que nombramos objetos. Todos los nombres en el espacio de nombres podrían tener atributos que pueden ser usado para buscar el objeto.

<sup>3</sup> JTS: Java Transaction Service.

Contenedor EJB: Maneja la ejecución de EJBs para las aplicaciones J2EE. EJBs y su contenedor corren en el servidor J2EE.

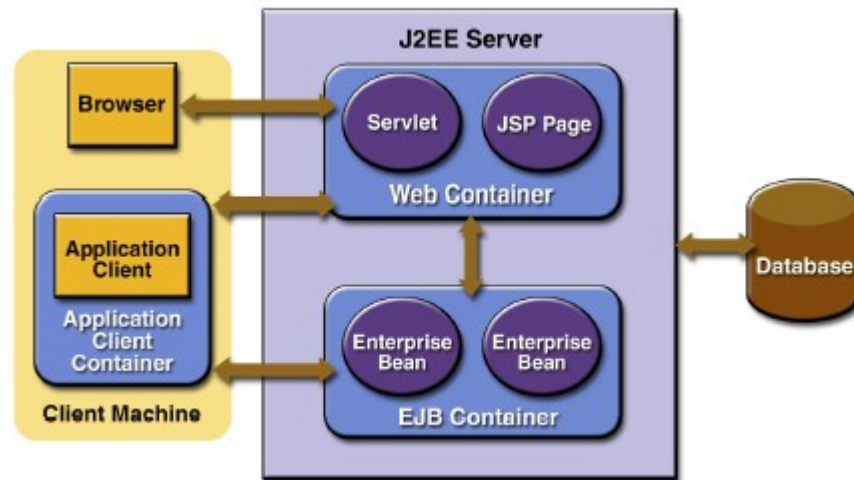
Contenedor WEB: Maneja la ejecución de páginas JSP y componentes servlets para aplicaciones J2EE. Estos componentes y su contenedor corren en el servidor J2EE.

Contenedor de aplicación cliente: Manejan la ejecución de componentes de aplicación clientes. Dichas aplicaciones y su contenedor corren en el cliente.

Contenedor Applet: Maneja la ejecución de APPLETS. Consiste de un navegador Web y java plug-ins que corren juntos en el cliente.

El proceso de despliegue instala componentes de aplicación J2EE en los contenedores J2EE como se ilustra en la figura 3.





**Figura 3: Servidor J2EE y sus contenedores [3]**

Como se puede observar, el servidor J2EE es el que proporciona los contenedores EJB y WEB, siendo este una porción “runtime” de un producto J2EE.

## **Demostración del uso de la especificación J2EE en el módulo StoreFront**

El módulo StoreFront tiene como tarea el proceso de gestión de las transacciones de compra/venta de productos en el sistema. Debido a su compleja funcionalidad, se realizó un análisis para determinar el diseño técnico del módulo.

El primer punto a evaluar fue la creación de los componentes Web y EJB. El componente Web contiene las páginas JSP y clases Java que permiten la comunicación con el componente EJB. El componente EJB almacena todos los Beans de sesión, los Beans de entidad y los Beans para mensajería necesarios para el módulo de acuerdo a la lógica del negocio.

Gracias a los APIs que contiene la especificación J2EE, se pudo hacer uso de la mayoría de ellos para determinadas partes del módulo, como se muestra en la siguiente tabla:

Funcionalidad	APIs utilizados
Catálogo	JSP, EJB, JNDI, JDBC.
Generación de Orden de Compra	JSP, EJB, JNDI
Comunicación a clientes	JMS, JavaMail

**Tabla 1: Funcionalidad de StoreFront y APIs de J2EE utilizados**

Estos APIs fueron suministrados por el contenedor correspondiente. Para el módulo se utilizó como contenedor Web a Apache Tomcat y como contenedor EJB a JBOSS. Así se cumple la presencia de los componentes en sus distintos contenedores lo que permite la separación adecuada para el desarrollo eficiente del módulo y su correspondiente despliegue, siguiendo las directrices de la especificación J2EE.

### **2.2.2MVC en Storefront**

## ¿Qué es MVC?

MVC es un patrón de diseño de vital importancia que se usa en la programación de aplicaciones interactivas que requieren de una interfaz de usuario flexible. [5] Está compuesto de 3 módulos diferentes llamados Modelo, Vista y Controlador, de allí su nombre (MVC).

El Modelo contiene los datos y la funcionalidad de la aplicación (lógica del negocio). En muchos casos involucra accesos a almacenamiento de datos. El equipo de desarrollo que maneja el modelo debe ser experto en escribir programas en DB2 COBOL, EJBs u otras tecnologías apropiadas para almacenar y manipular los datos de la empresa.

La Vista es la interfaz de usuario que muestra información sobre el modelo y que representa el dispositivo de entrada que se usa para modificarlo. El desarrollo de esta capa involucra el uso de páginas JSP y Beans de Java que almacenen los datos para ser usados por las páginas.

El Controlador es el código que determina el flujo general de la aplicación. Hace corresponder las peticiones que llegan del cliente con las acciones correspondientes y dirige las respuestas a las vistas adecuadas. Cada vista tiene un controlador asociado que puede ser una o más acciones de Struts<sup>1</sup>, archivos de configuración y servlets.

Capa	Responsabilidad	Colaboradores
Modelo	Contiene la funcionalidad de la aplicación. Lleva un registro de las vistas y controladores del sistema. Notifica los cambios en los datos a los componentes.	Vista Controlador
Controlador	Acepta los eventos de entrada. Traduce los eventos de entrada a peticiones al modelo o a las vistas. Implementa el procedimiento actualizar si es necesario.	Vista Modelo

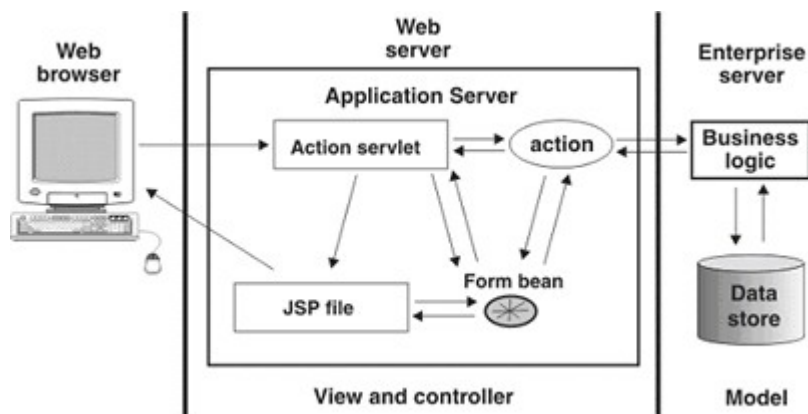
---

<sup>1</sup> Struts: Framework de programación código abierto para aplicaciones Web.

Vista	Crea e inicializa su controlador asociado. Muestra información al usuario. Actualiza la información. Recoge datos del modelo.	Controlador Modelo
-------	--	-----------------------

**Tabla 2 : Estructura del patrón de diseño MVC [5]**

La Figura 4 muestra un diagrama del patrón de diseño MVC, en el cual se observa al servidor Web en tiempo de ejecución que contiene los componentes de la vista y del controlador, mientras que una tercera capa, que usualmente está fuera del servidor Web, contiene el modelo.



**Figura 4: Diagrama del patrón de diseño MVC [6]**

### **Características**

Varias características son comunes para el patrón de diseño MVC. La interfaz de usuario es generalmente desarrollada con páginas JSP que no contienen ninguna lógica de negocio. Estas páginas representan los componentes de la vista de una arquitectura MVC.

Los formularios e hipervínculos en la interfaz de usuario que requieren lógica de negocio para ser ejecutadas se envían por una URI<sup>1</sup> de consulta que se mapea a un servlet activo. Este servlet recibe y procesa todos los pedidos que cambian el estado de una interacción del usuario con la aplicación. Este componente representa el componente controlador de una arquitectura MVC.

El “action servlet” selecciona e invoca una o más acciones para realizar la lógica de negocio pedida. Las acciones manipulan el estado de las interacciones de la aplicación con el usuario, típicamente creando o modificando Java Beans que se almacenan como pedidos o atributos de sesión (dependiendo en que medida se necesite que estén disponibles). Tales Java Beans representan los componentes del modelo de una arquitectura MVC. En lugar de producir la siguiente página de la interfaz de usuario directamente, las acciones generalmente utilizan las facilidades funcionales del método `RequestDispatcherforward()` de la

---

<sup>1</sup> URI: De sus siglas en inglés Uniform Resource Identifier.



API de Servlet para pasar el control a una página JSP apropiada para que reproduzca la siguiente página de la interfaz de usuario [6].

### **Beneficios**

El uso de la división del patrón de diseño MVC para el desarrollo de aplicaciones Web J2EE tiene varios beneficios: [6]

Permite distribuir el esfuerzo de desarrollo hasta cierto punto, tal que los cambios de implementación en una parte de la aplicación Web no requieran cambios en otros. Los responsables del desarrollo para escribir la lógica de negocio pueden trabajar independientemente de los responsables del desarrollo del flujo de control, de los diseñadores de páginas Web y a su vez estos también pueden trabajar en forma independiente.

Permite prototipar más fácilmente las tareas.

Permite migrar aplicaciones antiguas, porque la vista esta separada del modelo y del control y puede ser tolerable a diferentes plataformas y categorías de usuarios.

Puede mantener un ambiente que comprenda tecnologías diferentes a través de diferentes ubicaciones.

El diseño MVC tiene una estructura organizada que mejora la escalabilidad de soporte (construir aplicaciones más grandes) y es fácil de modificar y mantener (debido a la clara separación de tareas).

### **Diseño del módulo StoreFront**

Basándose en los conceptos del patrón MVC, se realizó una estructura del diseño del módulo que se implementó para el desarrollo.

El modelo estaría constituido por los diferentes Beans que conforman el corazón del módulo. En primera instancia estarán los Beans de entidad que mapean la base de datos. Cabe recalcar que se crearán Beans de entidad por cada tabla, por tanto servirán para todos los módulos del sistema.

Ya en la parte de la lógica del negocio, se crearán Beans de sesión que contengan funciones específicas y Beans manejados por mensajes para la comunicación con el usuario a través de mensajes.

La vista está compuesta de páginas JSP y páginas HTML.

### **2.2.3 JSP y Servlets en el StoreFront**

Para mayor comprensión y diferenciación de las dos tecnologías que se explicarán a continuación, se citarán por orden de aparición en el mercado.

### **Tecnología Java Servlet**

Tan pronto como el Web empezó a ser usado para entregar servicios, los proveedores de servicios reconocieron la necesidad del contenido dinámico. Los Applets, uno de los más tempranos intentos hacia este objetivo, se enfocaron a usar la plataforma cliente para entregar contenido dinámico a los usuarios. Al mismo tiempo, los desarrolladores también investigaron usando plataforma servidor para este propósito. Inicialmente, los scripts CGI (Common Gateway Interface) fueron la principal tecnología usada para generar contenido dinámico. Aunque ampliamente usados, la tecnología CGI tiene un número de defectos, incluyendo dependencia de plataforma y falta de escalabilidad. Para remediar estas limitaciones, la tecnología Java Servlet fue creada como una manera portable para proveer contenido dinámico orientado al usuario [3].

### **Definición de Servlet**

Un servlet es un componente Web manejado por un contenedor que genera contenido dinámico. En sí, es una clase de Java usada para

extender las capacidades de los servidores que albergan aplicaciones accedidas mediante un modelo de programación petición/respuesta (request-response).

Aunque los servlets pueden responder a cualquier tipo de requerimiento, son más comúnmente usados para extender las aplicaciones alojadas por servidores Web. Para tales aplicaciones, la tecnología Java Servlet define clases servlets HTTP específicas [3].

### **Características**

Entre sus características están:

Reemplazan a CGI, porque son más rápidos ya que usan un modelo de proceso diferente.

Poseen las ventajas que provee Java, por ejemplo independencia de plataforma, facilidad de uso, etc.

Producen HTML u otros contenidos en respuesta a un requerimiento HTTP<sup>1</sup>.

---

<sup>1</sup> HTTP: De sus siglas en inglés Hypertext Transfer Protocol

No tienen interfase gráfica.

Deben ajustarse al API de Servlets, que es un API estándar soportado por muchos servidores Web.

Además, pueden acceder a un gran conjunto de APIs disponibles en la plataforma Java.

### **Funciones**

Leer los datos enviados por el usuario. Típicamente enviados en un formulario en una página Web (datos explícitos), pero pueden venir también de un APPLET, de una aplicación cliente de HTTP (datos implícitos como de un request headers).

Procesan la información.

Genera resultados a través del cálculo directo de la respuesta o llamando a otro servidor (posiblemente remoto – accesos con RMI o CORBA), o accediendo a una base de datos, etc.

Recuperar información de un usuario embebida en la petición HTTP.

Formatear los resultados dentro de un documento, normalmente dentro de una página HTML.

Poner los parámetros de respuesta HTTP adecuados, pudiendo ser del tipo de documento devuelto (HTML), cookies, parámetros de cache, etc.

Devolver el documento al cliente a través de un documento textual (HTML), formato binario (GIF), comprimido (gzip), etc.

### **Origen y concepto de la tecnología Java Server Pages (JSP)**

La tecnología Java Server Pages (JSP) permite crear con facilidad contenido Web que contiene componentes dinámicos y estáticos [3].

Surge esta tecnología debido a ciertas dificultades que la tecnología Java Servlet tenía. Por ejemplo:



Con servlets es fácil leer datos de formularios, hacer peticiones y dar respuestas HTTP, usar cookies y variables de sesiones, compartir datos entre ellos, etc. Pero es una tarea difícil usar sentencias “println” para generar código HTML y mantener las páginas.

Nace entonces la idea de:

Usar código HTML para la mayoría de la página.

Marcar código servlet con etiquetas especiales.

Trasladar la página JSP entera dentro de un Servlet (una sola vez) y el servlet es el que será invocado.

Como resultado, JSP hace disponible todas las capacidades dinámicas de la tecnología Java Servlet pero provee un mayor acceso para crear contenido estático.

Las principales características de la tecnología JSP son:

Un lenguaje para desarrollar páginas JSP, las cuales son documentos basados en texto que describen cómo procesar un requerimiento (request) y construir una respuesta (response).

Un lenguaje de expresión para acceder a objetos del lado del servidor.

Mecanismos para definir extensiones para el lenguaje JSP.

La tecnología JSP además contiene un API que es usado por desarrolladores de contenedores Web.

### **Beneficios**

Aunque JSP técnicamente no puede hacer lo que Servlets tampoco hace, permite realizar lo siguiente:

Mantener el código HTML en las páginas, es decir leerlo y escribirlo.

Fácil para combinar templates estáticos, incluyendo HTML o XML.

Las páginas JSP compilan dinámicamente dentro de servlets.

Las etiquetas JSP (tags JSP) invocan a componentes JavaBeans™ .

Usa herramientas HTML para el desarrollo.

Permite la separación del código que crea el contenido del código HTML que presenta.

Facilita la separación de grupos de trabajo para el desarrollo de la aplicación, grupos para la creación de páginas y grupos para programación.

No es necesario que los desarrolladores de las páginas JSP conozcan programación en servlets, por lo tanto, facilita el desarrollo de páginas Web a los desarrolladores por su rápida comprensión.

### **Uso de JSP y Servlets en StoreFront de Eguana**

Debido a las facilidades que brinda a los desarrolladores la tecnología JSP, se la utilizó como base para las páginas de la vista en el módulo.

Entre las facilidades que se presentaron están la separación de código y el aprendizaje. Además se determinó evitar el uso de servlets debido a su compleja codificación.

Cabe acotar que a pesar de no haber utilizado la tecnología de servlets para el desarrollo, al momento de la compilación de las páginas JSP se crearán servlets que estarán almacenados en el servidor J2EE.

#### **2.2.4 Espectro completo de EJBs en el StoreFront**

Los “Enterprise Beans” son los componentes J2EE que implementan la tecnología Enterprise JavaBeans (EJB). Es una arquitectura de componentes del lado del servidor para desarrollar

aplicaciones de objetos distribuidos (multicapas), la cual encapsula la lógica de negocio de una aplicación. La lógica de negocio es el código que cumple el propósito de la aplicación. Normalmente están ubicados en la capa media de una aplicación.

Escritos en lenguaje Java, pueden estar compuestos por uno o más objetos, los cuales exponen todos sus servicios a través de una sola interfaz. Su definición y ambiente de ejecución está definido por una especificación.

Los Enterprise Beans corren en un contenedor EJB, un ambiente en tiempo de ejecución dentro del servidor J2EE. Aunque transparente al desarrollador de la aplicación, el contenedor EJB provee servicios de nivel de sistema tales como soporte transaccional y de persistencia. Estos servicios permiten rápidamente construir y desplegar Enterprise Beans, los cuales forman el corazón de las aplicaciones J2EE transaccionales.

## **Beneficios**

Por muchas razones, los Enterprise Beans simplifican el desarrollo de aplicaciones grandes y distribuidas. Primero, debido a que el contenedor EJB provee servicios de nivel de sistema a los Enterprise Beans, el desarrollador de los Beans puede concentrarse en resolver los problemas del negocio. El contenedor EJB (no el desarrollador de los Beans) es responsable de estos servicios tales como manejo de transacciones y autorización de seguridades.

Segundo, debido a que los Beans, y no los clientes, contienen la lógica del negocio, el desarrollador de la aplicación cliente puede enfocarse en la presentación. El desarrollador no tiene que codificar rutinas para implementar reglas de negocios o acceder a bases de datos. Como resultado, la aplicación cliente es ligera, un beneficio que es particularmente importante para clientes que corren en pequeños dispositivos.

Tercero, los Enterprise Beans son componentes portables, por tanto, el ensamblador de la aplicación puede construir nuevas aplicaciones de Beans existentes. Estas aplicaciones pueden correr en cualquier servidor J2EE compatible [3].

### **¿Cuándo usar Enterprise Beans?**

Se debe considerar el uso de Enterprise Beans si la aplicación cumple los siguientes requerimientos: [3]

Debe ser escalable. Para acomodar un número creciente de usuarios, se podría necesitar distribuir los componentes de la aplicación en múltiples máquinas. No solamente pueden los Enterprise Beans de una aplicación correr en diferentes máquinas, sino que su localidad permanecerá transparente a los clientes.

Transacciones son requeridas para asegurar integridad de datos. Los Enterprise Beans soportan transacciones, los mecanismos que manejan el acceso concurrente de objetos compartidos.

Tener una variedad de clientes. Con sólo unas pocas líneas de código, los clientes remotos pueden fácilmente localizar a los Enterprise Beans. Estos clientes pueden ser livianos y numerosos.

## **Tipos**

Los EJBs, como son útiles para modelar objetos de negocio dentro de una aplicación, poseen diferentes tipos de Beans, como se describe brevemente en la siguiente tabla:

<b>Tipo de Enterprise Bean</b>	<b>Propósito</b>
Session	Maneja una tarea para un cliente; implementa



	un servicio Web.
Entity	Representa un objeto de una entidad de negocio que existe en un almacenamiento persistente.
Message-Driven	Actúa como un “listener” para el API de JMS, procesando mensajes asincrónicos.

**Tabla 3: Sumario de tipos de Enterprise Beans [3]**

Según lo descrito, los tipos de Beans son:

Session Beans o Beans de Sesión.

Entity Beans o Beans de Entidad.

Message-Driven Beans o Beans dirigidos por mensajes.

### **Session Beans o Beans de Sesión**

Son los encargados de modelar los procesos del negocio, por lo que contienen toda la lógica correspondiente, como calcular valores, transferir información, acceder a otros sistemas, etc.

Representan sesiones interactivas con uno o más clientes. Los Beans de sesión pueden mantener un estado pero sólo durante el tiempo que

el cliente interactúa con el Bean. Esto significa que este tipo de Beans no almacenan sus datos en una base de datos después que el cliente termine el proceso. Por ello se suele decir que los Beans de sesión no son persistentes o son simplemente objetos en memoria.

Normalmente, cualquier llamada a un servicio del servidor debería comenzar con una llamada a un Bean de sesión. Ejemplos de Beans de sesión podrían ser un carrito de compras de una aplicación de negocio electrónico o un sistema verificador de tarjetas de crédito.

Los Beans de sesión no se comparten entre más de un cliente, sino que existe una correspondencia uno a uno entre Beans de sesión y clientes. Por esto, el contenedor EJB no necesita implementar mecanismos de manejo de concurrencia en el acceso a estos Beans.

Existen dos tipos de Beans de sesión:

Stateful Session Beans o Beans de sesión con estado.

Stateless Session Beans o Beans de sesión sin estado.

### **Stateless Session Beans o Beans de sesión sin estado**

Los Beans de sesión sin estado no se modifican con las llamadas de los clientes. Los métodos que ponen a disposición de las aplicaciones clientes son llamadas que reciben datos y devuelven resultados, pero que no modifican internamente el estado del Bean. Esta propiedad permite que el contenedor EJB pueda crear una reserva (pool) de instancias, todas ellas del mismo Bean de sesión sin estado y asignar cualquier instancia a cualquier cliente. Incluso un único Bean puede estar asignado a múltiples clientes, ya que la asignación sólo dura el tiempo de invocación del método solicitado por el cliente. Luego de cada invocación en contenedor puede optar por: destruir el Bean, recrearlo o inicializarlo. Estos normalmente deben recibir del cliente toda la información necesaria para realizar su tarea u obtenerla de una fuente externa como una base de datos.

El funcionamiento de los componentes EJB se basa fundamentalmente en el trabajo del contenedor EJB, el cual es un programa en Java que corre en el servidor y que contiene todas las clases y objetos necesarios para el correcto funcionamiento de los Enterprise Beans.

En la Figura 5 se puede ver una representación de alto nivel del funcionamiento básico de los Enterprise Beans. En primer lugar se puede ver que el cliente realiza peticiones al Bean y el servidor que contiene el Bean están ejecutándose en máquinas virtuales Java distintas, incluso pueden estar en distintos servidores. Hay que acotar que el cliente nunca se comunica directamente con el Enterprise Bean, sino que el contenedor EJB proporciona un objeto EJBObject que hace de interfaz. Cualquier petición del cliente (una llamada a un método de negocio del Enterprise Bean) se debe hacer a través del objeto EJB, el cual solicita al contenedor EJB una serie de servicios y se comunica con el Enterprise Bean. Por último, el Bean realiza las peticiones correspondientes a la base de datos.

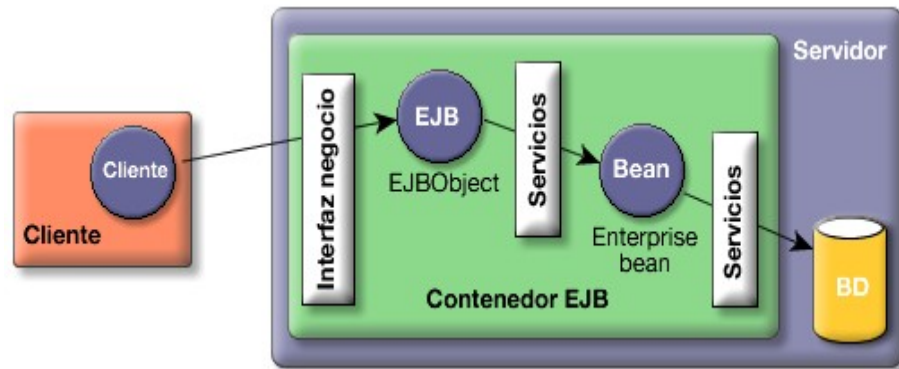


Figura 5: Funcionamiento de los Enterprise Beans [7]

Enterprise Beans en StoreFront de Eguana

Casi toda la gama de Enterprise Beans que existen en la actualidad en la especificación J2EE ha sido utilizada en el módulo StoreFront de Eguana.

Para llevar la lógica del negocio, como el carrito de compras, la búsqueda de productos en el catálogo y la generación de orden de compra se crearon Beans de sesión con estado.

El mapeo del motor de EJB con la base de datos se realizó utilizando Beans de entidad, cuya persistencia estaría a cargo del contenedor EJB, lo que garantizaba un menor grado de error.

Por último, debido a la funcionalidad del módulo, era indispensable el manejo de Beans dirigidos por mensajes para la comunicación entre el usuario y el sistema en los pasos finales del proceso de compra. Estos Beans se encargarían de mantener informado a usuarios vendedores de estado de sus ventas asignadas, mantener comunicación con el banco y

notificar la aprobación correspondiente para el pago, permisos para despacho de los productos, etc.

En sí, el corazón del módulo está compuesto por Enterprise Beans lo cual permite al mismo tener todas las bondades de una completa aplicación empresarial Web.

#### **2.2.5 Manejo de transacciones en proceso de compra**

Una aplicación empresarial típica lee y almacena información en una o más bases de datos. Debido a que esta información es crítica para las operaciones del negocio, ésta tiene que ser exacta, actual y confiable. La integridad de los datos se perdería si múltiples programas actualizaran la misma información simultáneamente. Además se perdería si un sistema ha fallado mientras procesaba una transacción comercial y sólo una parte de la información afectada fue actualizada. Para prevenir ambos escenarios, las transacciones aseguran integridad



de datos. Las transacciones controlan el acceso concurrente de datos por múltiples programas. En el caso de una falla del sistema, las transacciones aseguran que después de la recuperación, los datos estarán en un estado consistente. [3]

### **¿Qué es una transacción? [3]**

Para emular una transacción de negocios, un programa puede necesitar realizar varios pasos. Por ejemplo, un programa financiero podría transferir los fondos de una cuenta corriente a una cuenta del ahorro con los pasos listada en el seudo código siguiente:

```
begin transaction
```

```
    debitar cuenta corriente
```

```
    acreditar cuenta ahorros
```

```
    actualizar historial
```

```
commit transaction
```

Como se observa, se realizan los tres pasos o ninguno, de lo contrario la integridad de datos se pierde. Debido a que los pasos dentro de una transacción son un todo unificado, una transacción se define a menudo como una unidad indivisible de trabajo.

Una transacción puede finalizar de dos maneras: con un “commit” o un “rollback”. Cuando una transacción realiza un “commit”, las modificaciones de los datos hechas por sus sentencias son almacenadas. Si una sentencia dentro de una transacción falla, la transacción realiza un “rollback”, deshaciendo los efectos de todas las sentencias en la transacción.

En el ejemplo, las sentencias “commit” y “rollback” marcan los límites de la transacción. Cuando se diseña un Enterprise Bean, se marcan los límites de dos formas:

Manejadas por el contenedor

Manejadas por el Bean

**Transacciones manejadas por el contenedor.-** En un Enterprise Bean con transacciones manejadas por el contenedor, el contenedor EJB es el que marca los límites de la transacción. Este tipo de transacciones se puede usar con cualquier tipo de Enterprise Bean: de sesión, de entidad o manejado por mensajes. Además, simplifican el desarrollo porque el código del Enterprise Bean no marca explícitamente los límites de la transacción. El código no incluye sentencias que comienzan o finalizan la transacción.

Típicamente, el contenedor comienza una transacción inmediatamente antes de comenzar un método del Enterprise Bean. Este realiza el “commit” de la transacción sólo antes de salir del método. Cada método puede ser asociado con una transacción simple. Transacciones múltiples o anidadas no son permitidas dentro de un método.

Las transacciones manejadas por el contenedor no requieren que todos los métodos sean asociados con transacciones. Cuando se despliega un Bean, se especifica cuál de los métodos del Bean serán asociados con transacciones especificando los atributos de una transacción. Esto se realiza en un archivo conocido como “deployment descriptor” de extensión XML propios del contenedor.

**Transacciones manejadas por el Bean.-** En una transacción manejada por el Bean, el código en el Bean de sesión o en el Bean manejado por mensajes explícitamente marcan los límites de la transacción. Un Bean de entidad no puede tener transacciones manejadas por Bean; éste debe usar en su lugar transacciones manejadas por el contenedor.

Aunque los Beans con transacciones manejadas por el contenedor requieren menos código, éstos tienen una limitación: cuando un método es ejecutado, éste puede ser asociado a una transacción o a ninguna.

Si esta limitación hace difícil codificar el Bean, se debe considerar las transacciones manejadas por Bean, las cuales permiten incluir sentencias de “commit” o “rollback” a necesidad del desarrollador.

Cuando se codifica este tipo de transacciones para Beans de sesión o de mensajes, se debe decidir por usar transacciones JDBC o JTA.

### **El uso de transacciones en el StoreFront de Eguana**

Debido a la facilidad que brinda la especificación J2EE para el caso de transacciones, específicamente en el hecho de dejar en libertad su manejo en una aplicación Web multicapa, se procedió a encargar al contenedor EJB esta tarea. Esto permitió que las transacciones para todos los Enterprise Beans del módulo tengan un mismo concepto de creación, es decir antes de iniciar cualquier método de los Beans y, que sean configuradas a nivel de los archivos de configuración del contenedor.

## 2.3 Herramientas de código abierto en el proyecto

Código abierto o “Open Source” como se lo conoce en inglés es un programa o software cuyo código fuente está disponible de manera gratuita al público, el cual ha sido desarrollado con la colaboración de una comunidad de programadores voluntarios. Como consecuencia los programas de código abierto son gratis y el público tiene la potestad de modificarlos y redistribuirlos siempre y cuando mantenga el esquema de código abierto y use la misma licencia de distribución del software original.

El movimiento de código abierto se contrasta con el software propietario, el cual suele ser muy costoso y mantiene su código fuente secreto. En los últimos años este movimiento ha logrado adquirir una gran aceptación por parte de los desarrolladores y el público en general.

El atractivo para el público es obviamente su costo, gratis. Para los

programadores su atractivo es una mezcla de idealismo en la forma de rechazo al alto precio y monopolio generado por las grandes corporaciones desarrolladoras de software y además la oportunidad de colaborar libremente con sus colegas. Sería falso argumentar que el desarrollo de código abierto no trae ganancias monetarias como lo demuestra la creación de diversas compañías que basan sus productos en código abierto como lo son Red Hat, JBOSS, o Palo Santo. En este esquema de trabajo las ganancias están principalmente en el soporte al usuario final del software.

El sistema Eguana es desarrollado en herramientas de programación de código abierto por lo tanto Eguana es código abierto. Además de ser desarrollado en herramientas de código abierto, Eguana es ejecutado en un ambiente de código abierto. En esta sección se describirá las herramientas de desarrollo que usamos en este proyecto, Eclipse y Sofía, y el software de ambiente en el cuál se ejecuta Eguana: Linux, MySQL, JBOSS.

### **2.3.1 Linux**

“Linux es un sistema operativo gratis tipo UNIX creado originalmente por Linus Torvalds con la ayuda de varios desarrolladores alrededor del mundo. Desarrollado bajo la licencia GNU (General Public License), el código fuente de Linux es gratis y disponible libremente para cualquiera que desee obtenerlo” [8]

Linux es un sistema operativo código abierto orientado al uso en servidores. Este sistema operativo ha demostrado ser en los últimos años una excelente alternativa a sistemas propietarios como UNIX o Windows. Existen varias versiones o distribuciones de Linux. Una distribución de Linux es una compilación y adaptación específica de su código fuente hecha por una comunidad de programadores o por una compañía. Existe un extenso debate sobre cuál distribución es mejor y hasta ahora no se ha llegado a ningún consenso, lo cierto es que



existen unas más populares que otras. Las más populares en el mercado son: Red Hat, Mandrake, Debian, Suse. Para el proyecto Eguana se eligió Red Hat debido a que esta es la más popular en nuestro país.

Dentro de Red Hat también existen varias versiones de su distribución. Hasta el momento de escribir este documento, Eguana ha sido probado con éxito en las versiones: Red Hat 8, Red Hat 9, Fedora Core 1, Fedora Core 2 y Fedora Core 3.

### **2.3.2JBOSS**

“JBOSS es el servidor de aplicaciones código abierto J2EE líder en el mercado, provee una plataforma de alto rendimiento para aplicaciones e-bussiness” [9]

JBOSS es sin duda la implementación de un servidor J2EE más popular en el mercado. Es código abierto, usa la licencia LGPL<sup>1</sup> y por lo tanto su uso es gratuito. En Eguana decidimos usar la versión 3.2.4 aunque al momento de escritura de este documento JBOSS se encuentra ya en su versión 4. La decisión de usar JBOSS como servidor de aplicaciones en Eguana fue la que más impacto causó en el desarrollo de la aplicación debido a que este importante componente del proyecto determina la arquitectura general del sistema.

### **2.3.2.1 Características del JBOSS**

JBOSS es lo que se define también como Middleware, software que se encuentra entre la aplicación y la base de datos. JBOSS es el encargado de proveer servicios a la aplicación para que ésta pueda funcionar. Su funcionamiento interno lo hace en forma de capas como se muestra en la Figura 6 a continuación:

---

<sup>1</sup> LGPL: De sus siglas en inglés Lesser General Public License

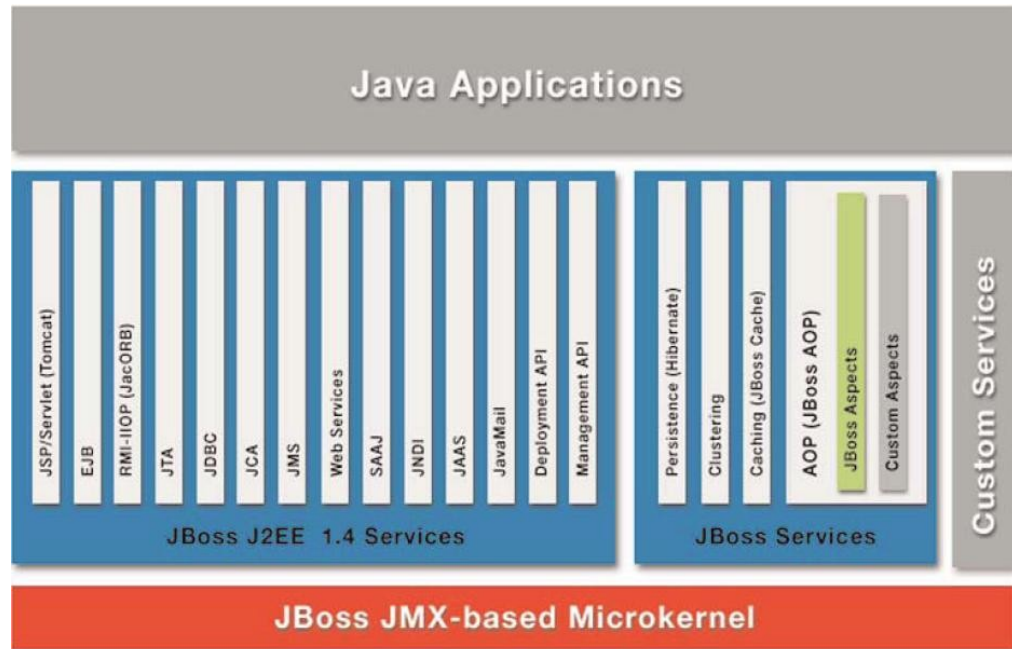


Figura 6: Diagrama de capas de JBOSS [10]

En la capa más baja tenemos el microkernel basado en JMX y el cuál se encarga de gestionar todas las tareas y servicios que el sistema provee de una manera ordenada.

En la siguiente capa tenemos los servicios de JBOSS, estos servicios pueden ser tanto los servicios que provee J2EE en general como servicios propietarios de JBOSS. Además JBOSS añade la flexibilidad de servicios personalizados, esto es servicios hechos por el desarrollador de la aplicación o por terceros. Esta es la capa más importante del servidor de aplicaciones, es aquí donde se aplican las pautas de J2EE y cumple su estándar, es aquí también donde JBOSS adquiere su identidad y se lo diferencia de otros servidores de aplicaciones y finalmente es aquí donde el desarrollador saca provecho máximo al poder de este software.

La última capa, la capa superior, es donde se encuentra la aplicación.

Aquí es donde la aplicación desarrollada por el equipo de programación

llama a los servicios de JBOSS para aspectos que van desde mensajería hasta persistencia [9].

### **2.3.2.2Ventajas**

Existe una enorme cantidad de servidores de aplicaciones en el mercado. Prácticamente cada compañía cuyo negocio base es la Tecnología de Información ha desarrollado su propio servidor de aplicaciones. La numerosa población de este tipo de software ha hecho posible destacar las ventajas que JBOSS tiene ante sus competidores.

Existen muchos criterios de evaluación para un servidor de aplicaciones, entre los más importantes tenemos:

1. Rendimiento
2. Precio
3. Estándar

4. Portabilidad

5. Popularidad

De estos cinco criterios JBOSS mantiene una gran ventaja sobre la mayoría de sus competidores.

**Rendimiento:** Es justo afirmar que en este aspecto JBOSS se mantiene a la par o con ligera ventaja con la mayoría de sus competidores. La base de JBOSS es Java, una tecnología conocida por su gran consumo de recursos. Nuevas tecnologías incorporadas a Java (Hot Spot, JIT) y JBOSS (Microkernel) han permitido eliminar estas desventajas.

**Precio:** Este criterio es el fuerte de JBOSS, simplemente es gratis. Su licencia LGPL requiere que se uso sea gratuito sin importar la versión. Esto se contrasta con muchos de sus competidores los cuáles requieren licencias en miles e incluso decenas de miles de dólares.

**Estándar:** JBOSS tiene licencia J2EE y soporta, en su última versión, el estándar J2EE 1.4. La mayoría de servidores de aplicaciones soportan hasta J2EE 1.3 y no tienen licencia J2EE. La importancia del estándar radica en que la tecnología usada en el servidor de aplicaciones no está amarrada a ninguna compañía en particular lo cual provee una mayor libertad y seguridad al desarrollador.

**Portabilidad:** JBOSS es independiente de cualquier sistema operativo, solo necesita el JDK 1.3 o versión más alta para funcionar.

**Popularidad:** Este es un valor difícil de medir con exactitud pero fácil de apreciar. Debido a las ventajas arriba descritas la popularidad de JBOSS está en aumento y es actualmente mayor a la mayoría de sus competidores. La popularidad es una ventaja muy importante para el desarrollador ya que provee una mayor comunidad de soporte.

### 2.3.2.3 Esquemas de seguridad del JBOSS

La seguridad es una parte fundamental de cualquier aplicación empresarial. Es necesario restringir quien esta permitido a acceder la aplicación y controlar las operaciones que los usuarios puedan ejecutar. La especificación J2EE define un simple modelo de seguridad para los EJBs y los componentes Web basado en roles. El componente de JBOSS que maneja la seguridad es JBossSX. La seguridad de extensión JBossSX provee soporte para la seguridad declarativa basada en roles del modelo J2EE como también la integración de seguridad especializada a través de una capa de seguridad Proxy. La implementación original del modelo J2EE de seguridad declarativa es basado en los módulos JAAS<sup>1</sup> [11].

La seguridad en JBOSS, como indica el modelo J2EE, está basada en roles. Es altamente configurable tiene un gran poder y alcance. En

---

<sup>1</sup> JAAS: Java Authentication and Authorization Service, es un conjunto de APIs usado para autenticación y autorización.



Eguana usamos el sistema de roles para definir cuales son las acciones que cada usuario puede ejercer. Los roles se definen en los descriptores de despliegue de cada modulo Web de la aplicación. Aquí también se definen los dominios de acceso para cada rol y los métodos de autenticación de los usuarios.

Además de proveer una manera segura de restringir el acceso a los usuarios, JBOSS también provee en el mismo módulo de seguridad formas confiables de mantener la privacidad e integridad de los datos mediante el uso de encriptación SSL.

JBoss, mediante su modulo JBossSX, provee diversas opciones y herramientas de seguridad para el diseñador de aplicaciones. La seguridad es un factor vital de toda aplicación Web y esta hasta el momento de escritura de este documento ha sido inexpugnable en JBoss.

### 2.3.3MySQL

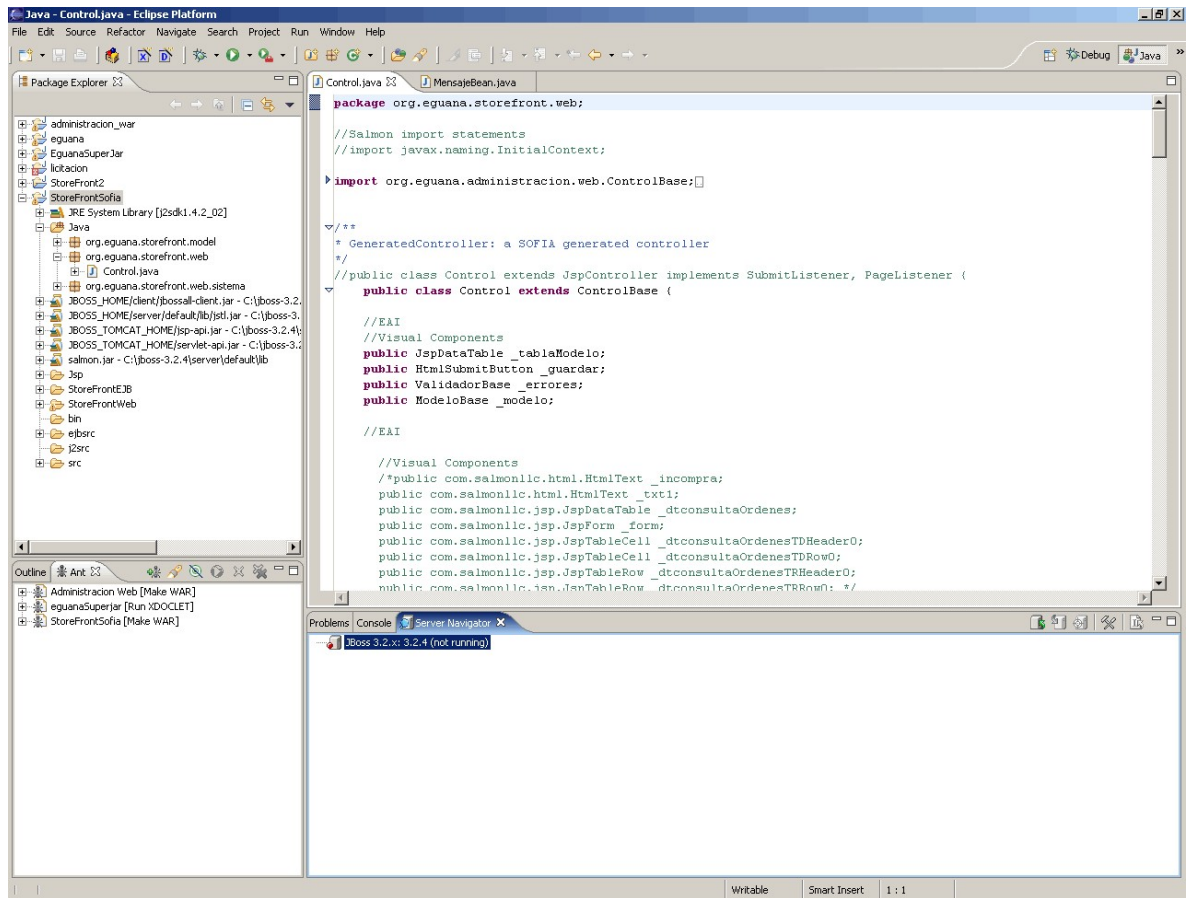
“El servidor de base de datos MySQL es la base de datos de código abierto más popular del mundo... Su arquitectura lo hace extremadamente rápido y fácil de configurar. Extensivo reuso de código y una filosofía minimalista en el diseño de características da como resultado una base de datos compacto, extremadamente rápida, estable y fácil de desplegar.” [12]

MySQL es la base de datos de código más popular en el mercado. A pesar de no poseer muchas características estándares en otras bases de datos, como por ejemplo integridad referencial, MySQL se suscribe a la filosofía de la simplicidad. Muchas de las tareas de mantenimiento e integridad de datos las deja en las manos del desarrollador o del servidor de aplicaciones y se preocupa solamente en hacer bien las tareas específicas de una base de datos. Esta filosofía de desarrollo ha

permitido que MySQL sea una base datos extremadamente simple y fácil de usar.

### **2.3.4Eclipse**

“Eclipse es una plataforma de integración de herramientas construida por una comunidad abierta de proveedores de herramientas. Operando dentro del paradigma de código abierto, con una licencia de uso público común que provee código fuente gratis y derechos de distribución mundial, la plataforma Eclipse provee a desarrolladores de herramientas un nivel inigualable de flexibilidad y control sobre sus tecnologías de software.” [13]



### **Figura 7: Pantalla de Eclipse**

Originalmente un producto de IBM, decir que Eclipse es simplemente un IDE (Integrated Development Enviroment) o ambiente de programación es subestimar el poder completo de esta herramienta. Una mejor forma de describirlo es como una plataforma donde se pueden integrar múltiples IDEs. Eclipse no esta limitado a ninguna tecnología de desarrollo, está abierto, mediante una arquitectura de plug-ins, a recibir funcionalidad para cualquier tecnología de cualquier proveedor. El hecho de que es código abierto, su increíble flexibilidad, y su excelente reputación convirtieron a Eclipse la primera y única opción como la herramienta de desarrollo del proyecto Eguana. Actualmente Eclipse está en su versión 3.0 la cuál utilizamos en el proyecto.

#### **2.3.5 Sofia Framework**

Sofia Framework, de Salmon LLC, es una herramienta para desarrollar aplicaciones Web de principio a fin en el ciclo de desarrollo.

Esta basado conceptualmente en el MVC definido por J2EE y es muy parecido a otras herramientas MVC código abierto como por ejemplo Apache Struts. Sofia se diferencia de la competencia en el hecho de que su alcance va mucho más allá de satisfacer el estandar MVC J2EE. Sofia provee una integración completa con las mejores herramientas de desarrollo en el mercado y provee una librería robusta de clases J2EE y tags JSP [14].

Con Sofia los desarrolladores pueden construir aplicaciones rápidamente usando la tecnología de Servlets/JSP. Sofia se integra a herramientas como Dreamweaver y Eclipse mediante módulos, plugins y extensiones. Las librerías de clase JSP proveen una amplia gama de elementos visuales como botones, menús, barras de navegación etc.

Sofia se encarga del “trabajo sucio” de la programación MVC permitiendo un aumento de productividad, un esquema consistente de programación, y enfocar al desarrollador en la lógica de negocio de la aplicación.

#### **2.3.5.1 Ventajas del modelo MVC**

Las mayores ventajas de usar Sofia son su capacidad de acortar el tiempo de desarrollo y obligar al desarrollador a usar el modelo MVC. El modelo MVC, como se explico en la sección 2.2.2 tiene muchas ventajas.

MVC resuelve el problema generado cuando una aplicación necesita ser utilizada de manera diferente en varios tipos de vistas e interfaces. Esto obligaba al desarrollador a repetir código en cada vista o interfase. Este esquema es muy ineficiente e inevitablemente errores e inconsistencias aparecerán en la duplicación de código.

MVC permite separar la vista o interfase de la lógica del negocio, los datos, y los controles de la interfase. Al tener aislado estas críticas secciones de la aplicación podemos fácilmente reciclar código, centralizar control, acelerar el proceso de desarrollo y construir una aplicación fácil de mantener, robusta y eficiente.

### **2.3.5.2J2EE y MVC**



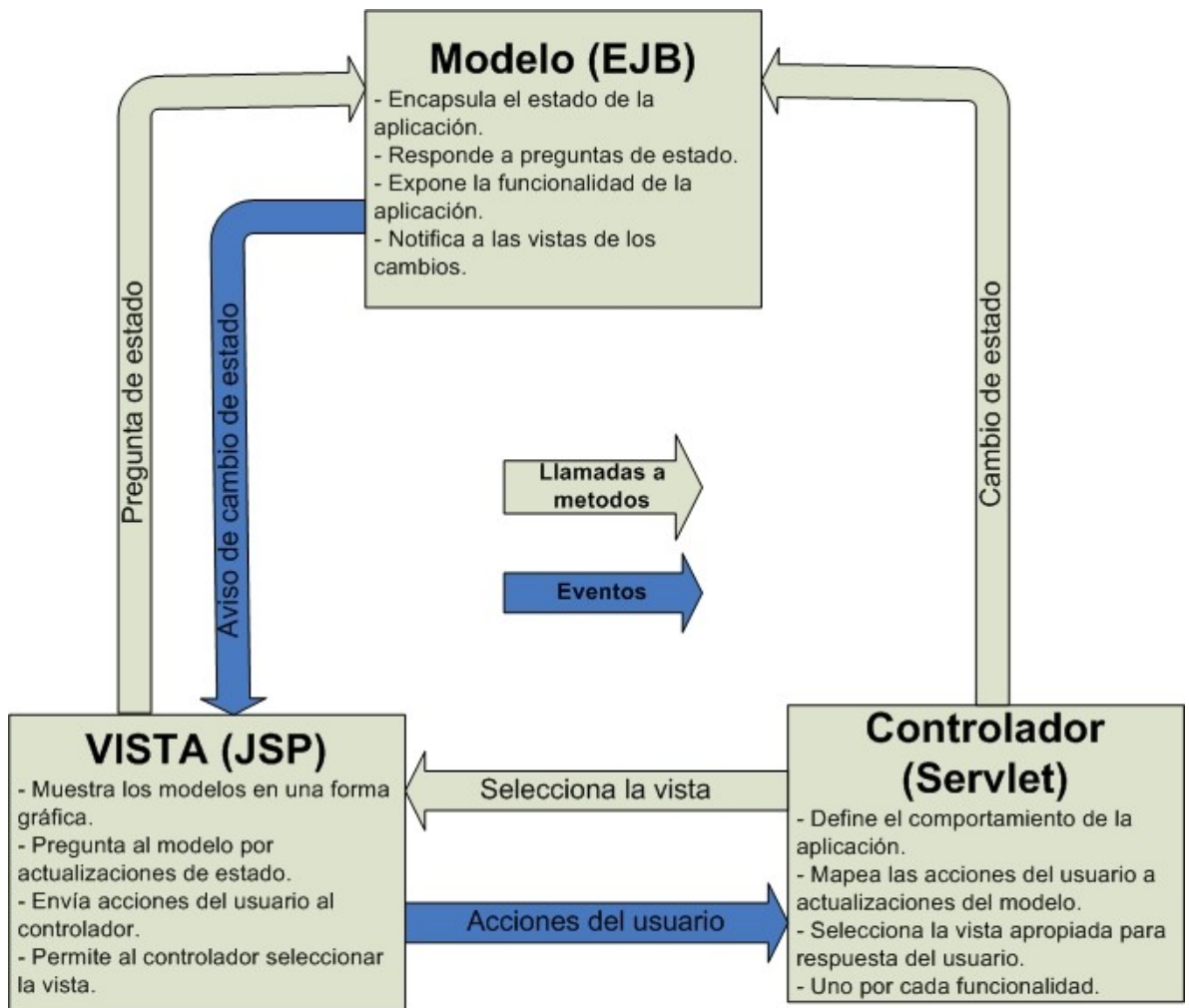
El estándar J2EE define el modelo MVC como parte principal de la filosofía de su tecnología. J2EE recomienda usar al pie de la letra el modelo MVC para obtener un mayor provecho de la plataforma de desarrollo Java.

Existen varias estrategias en J2EE para aplicar esta tecnología:

Cientes Web: JSP como vista. Servlets como controladores. EJBs como modelo.

Controlador centralizado: En lugar de tener varios Servlets como controladores se puede tener uno solo el cuál sería más manejable.

A continuación se muestra un esquema de definición del modelo MVC en el paradigma J2EE (Figura 8).



**Figura 8: Diagrama MVC (Modelo Vista Controlador) [15]**

### **2.3.6 Integración Linux-MySQL-JBOSS**

Linux, MySQL, y JBOSS forman un eje en el cuál se desenvuelve el ambiente de ejecución de Eguana. Para que Eguana pueda funcionar de manera rápida y eficiente en un ambiente de producción es necesario que estos tres componentes estén perfectamente integrados.

Linux, siendo el sistema operativo elegido para el funcionamiento, tiene que cumplir ciertos requisitos para poder integrarse con MySQL y JBOSS. El primer y más importante requisito es la incorporación de J2SDK al sistema operativo. La última versión de Java debe de estar instalada correctamente, todas las variables de ambiente que éste requiera también deben de estar definidas correctamente en el sistema. El segundo requisito es que el sistema operativo pueda dar las suficientes prestaciones a MySQL y JBOSS para funcionar de una

manera eficiente. Esto quiere decir que Linux no debe de estar sobrecargado con otros servicios que consuman recursos o causen conflictos de operación con JBOSS o MySQL. JBOSS requiere una gran cantidad de memoria y un gran número de puertos de red libres para poder funcionar. Los siguientes paquetes de software pueden potencialmente interferir con la aplicación: Apache, Tomcat, Oracle, Postgres, y versiones anteriores de MySQL o JBOSS en el mismo sistema.

Linux es sensible a la capitalización de los caracteres por lo tanto la versión de MySQL para Linux también lo es. Un error común al traspasar código desarrollado en herramientas ejecutadas en Windows a Linux es olvidar este aspecto. Es muy importante que el nombre de las tablas y objetos se mantengan igual durante todo el desarrollo del proyecto y a través de todos los módulos. En Windows la tabla COMPRAS y la tabla Compras se refieren a la misma tabla, en Linux no. Es importante además que MySQL y JBOSS se ejecuten e instalen como un servicio en el sistema operativo Linux.

JBOSS intenta mantener en lo más posible la sensibilidad a la capitalización mediante Java o mediante su propia funcionalidad. JBOSS 3.2.4 requiere por lo menos al JSDK 1.3 o mayor. Además JBOSS tiene que tener bien configuradas algunas variables en el sistema operativo, ejemplo JBOSS\_HOME. JBOSS necesita un driver JDBC para conectarse con MySQL. Existen varias implementaciones de este driver. En nuestro proyecto usamos la clase `org.gjt.mm.mysql.Driver` en el jar MySQL Connector versión 3.0.

En resumen JBOSS y MySQL están diseñados para integrarse con Linux sin ningún cambio grande en la configuración de ambos paquetes de software. JBOSS corre sobre Java el cuál es independiente de plataforma. La integración en nuestro proyecto es eficiente y prácticamente transparente.

### **2.3.7 Justificativo del uso de las herramientas**

Debido a la popularidad del movimiento de código de abierto existe una inmensa cantidad de programas disponibles para escoger en nuestro proyecto. Por esta razón es necesario justificar la elección de cada una de estas herramientas o programas.

Red Hat Linux: De todos los sistemas operativos en código abierto como por ejemplo FreeBSD y OpenSolaris, Linux es sin duda el más popular y de todas las distribuciones de Linux, Red Hat es la más popular en el Ecuador. El hecho que un producto sea más popular que otro no significa que sea mejor, muchos argumentan que FreeBSD es más robusto que Linux. Se escogió el producto más popular porque la popularidad si implica un mayor banco de información de soporte en el Internet, en la literatura técnica, y en personal técnico. Además presenta un ambiente de desarrollo más realista en nuestro medio.

JBOSS: Las razones descritas en la sección 2.3.2.2 y el hecho de que JBOSS es código abierto es razón suficiente para justificar su uso. A continuación mostramos una tabla de comparación de JBOSS con algunos de sus competidores.



Producto	Licencia J2EE	Certificado J2EE	Precio	Plataforma
JBOSS	Sí	1.4	Gratis	Cualquier plataforma con JDK 1.3+
Websphere	Sí	1.3	\$12,000	NT, Win2K, Solaris,AIX, OS/400, HP-UX, Red Hat Linux, SUSE Linux, Turbo Linux, Linux/390, NetWare, OS/390.
Oracle	Sí	1.3	\$20,000	Solaris, HP-UX, Redhat Linux, United Linux
Resin	NO	1.4	\$500	Cualquier plataforma con JDK 1.3+
Jonas	Sí	1.4	Gratis	NT, Linux, Solaris, AIX, HP-UX, Win2K, Netware
JRun	Sí	1.3	\$899	NT, Win2K, WinXP, Solaris, SUSE Linux, Red Hat Linux, HP-UX, Compaq Tru64, AIX

**Tabla 4: Comparación JBOSS y competencia [16]**

**MySQL:** Existen varias bases de datos código abierto, entre ellas están PostgreSQL, y Hypersonic. MySQL, además de ser popular, ofrece un excelente compromiso entre simplicidad y robustez. Hypersonic es muy simple y por lo tanto inapropiado para una base de datos muy grande.

PostgreSQL a pesar de ser muy robusto puede volverse complicado de mantener. El resto de bases de datos código abierto no han alcanzado aún una madurez de desarrollo aceptable.

**Eclipse:** Hasta el momento de escritura de este documento es imposible encontrar una herramienta de desarrollo código abierto que alcance la calidad de Eclipse. Eclipse es sin duda varias ordenes de magnitud superior a sus competidores de código abierto e incluso a muchos competidores de código propietario.

**Sofia Framework:** Esta herramienta permite acelerar el proceso de desarrollo y obliga al programador a usar de una manera clara las bondades del modelo MVC. Sofia provee integración en todas las etapas de desarrollo de una aplicación J2EE, desde la interfase gráfica con extensiones en Dreamweaver hasta la base de datos con gestores de conexión. Esta funcionalidad lleva a Sofia más allá del modelo MVC

donde la mayoría de sus competidores código abierto se quedan. No existe una herramienta MVC código abierto más completa que Sofia.

# CAPÍTULO 3

## ***3. Implementación del módulo storefront de eguana***

### **3.1 Análisis**

#### ***Análisis de costos***

En esta sección analizaremos los costos asociados con la implementación de Eguana para la empresa que lo administra y comercializa. Los costos de comercialización (ejemplo: publicidad, marketing, etc.) no se incluyen debido a que están fuera del alcance de este documento. Los costos de implementación se pueden dividir en dos partes: desarrollo e infraestructura. Los costos de desarrollo se

basaran en un estimado de horas-hombre. Los costos de infraestructura asumen que la empresa que implementará Eguana ya posee un edificio físico con computadoras interconectadas en una red de datos y con acceso a Internet.

**Desarrollo.-** De acuerdo a la experiencia obtenida en el desarrollo de este software podemos afirmar que un equipo de 9 personas trabajando 40 horas semanales durante cinco meses tiene tiempo suficiente para terminar el proyecto. Asumiendo un costo promedio de \$4 por hombre-hora para tres líderes de proyecto y \$2.5 para programadores tenemos la siguiente tabla:

No. Recurso Humano	# horas por semana	# semanas del proyecto	Costo por hora	Total costo
6	40	20	\$2.5	\$12,000
3 (líderes proyecto)	40	20	\$4	\$9,600
Total				\$21,600

**Tabla 5: Análisis horas - hombres**

Debido a que usaremos solamente herramientas de código abierto para el desarrollo de este proyecto, el costo de las licencias de desarrollo es cero. Por lo tanto el total de desarrollo es:

Total costo desarrolladores + total costo licencias = \$21,600 + \$ 0 =  
\$21,600

**Infraestructura.-** Infraestructura implica los servidores y equipos de apoyo además de las licencias de software de servidor. Las aplicaciones del servidor junto con el sistema operativo de este es también de código abierto por lo tanto su costo es cero. Se estima que para empezar exitosamente y dar un servicio de calidad es suficiente un servidor con las siguientes características:

Dos procesadores Intel Xeon de 3Ghz.

2 Gigabytes de memoria RAM.

6 discos SCSI Hot Swap en RAID 5 de 36

Gigabytes cada uno. 180 Gb total.

1 Tarjeta Ethernet

Unidad de respaldo de cintas SCSI.

A continuación tenemos la siguiente tabla:

<b>Costo estimado del servidor</b>	\$7,000
<b>Costo de licencias en el servidor</b>	0
<b>Total</b>	\$7,000

**Tabla 6: Análisis de costo**

Con estos resultados obtenemos que el total del costo de implementación de Eguana es:

Costo de desarrollo + Costo de infraestructura = \$21,600 + \$7,000 =  
\$28,600

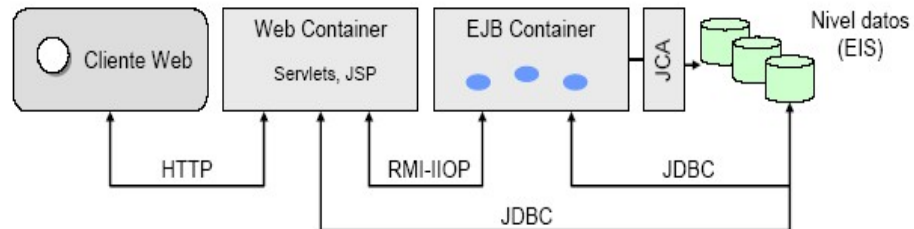
Como podemos concluir en este análisis de costo, Eguana es un proyecto relativamente barato tomando en cuenta su escala.

### ***Definición de la Arquitectura del sistema***



La implementación del sistema está basada en una arquitectura de n capas con tecnología EJB, la misma que permite que cualquier capa sea modificada sin afectar la funcionalidad del resto de las otras capas; esto mejora y facilita el mantenimiento e incrementa la reutilización del código. Obedece al patrón de diseño MVC propuesto por la tecnología J2EE, tal como fue explicado anteriormente. Este patrón está conformado por 3 capas: la Capa Vista, la Capa Modelo y la Capa Controlador. La Capa Vista se implementó mediante el uso de páginas HTML y JSP y la utilidad del FrameWork Sofía, y su interacción con la capa controlador y la capa modelo se realizó mediante el uso de Java RMI o RMI-IIOP. En cuanto a la Capa del Modelo, ésta hará uso de la tecnología Java Database Connectivity (JDBC) y la arquitectura EJB para comunicarse con el manejador de base de datos relacional.

La siguiente figura presenta una idea de la arquitectura de la aplicación Eguana:



**Figura 9: Diagrama arquitectura Eguana con tecnología J2EE basada en Web [23]**

## ***Metodologías de Desarrollo***

Internet ha traído consigo necesidades nuevas, que ha hecho que las necesidades de desarrollo también deban cambiar. Si hasta ahora ha existido un cambio en los lenguajes de desarrollo utilizados, debemos adaptar todavía los demás puntos que forman parte del

desarrollo de un proyecto, como pueden ser gestión y análisis. Debemos adaptar nuestras metodologías a las necesidades de estos proyectos.

En todo análisis y diseño de software es indispensable la utilización de una metodología de desarrollo que permita definir de manera estándar las necesidades del proyecto a implementar. La metodología usada para el desarrollo del proyecto es el de Paradigma Orientado a Objetos usando una base de datos relacional

Para la documentación del diseño del sistema hemos seleccionado el Lenguaje de Modelamiento Unificado (UML), el cual es seguido por las herramientas más comunes de diseño, y por ahora es un estándar universal que han adoptado todos los fabricantes de software.

El Lenguaje de Modelamiento Unificado (UML - Unified Modeling Language) es una especificación de notación orientada a objetos,

también definida como un lenguaje gráfico para visualizar, especificar y documentar cada una de las partes que comprende el desarrollo de software. UML entrega una forma de modelar cosas conceptuales como lo son procesos de negocio y funciones de sistema, además de cosas concretas como los componentes de software reusables [17].

Para la documentación de la base de datos se usó diagramas entidad – relación.

### **Diagramas UML**

Son la esencia de UML. Cada diagrama usa la anotación pertinente y la suma de estos diagramas crean las diferentes vistas del proyecto. Los diagramas a representar dependerán del sistema a

desarrollar, para el diseño de Eguana se usaron específicamente los siguientes diagramas:

Casos de uso.

Diagrama de Estado.

Diagrama de clases.

Diagrama de interacción de Objetos.

### **Tecnologías Complementarias**

En el diseño de un software es importante el lenguaje y las herramientas con las cuales se lo implementarán. Si bien tomamos en cuenta el rendimiento de las distintas soluciones a la hora de diseñar nuestro sistema, hubieron varios puntos en los cuales la exploración de las posibilidades de ciertas tecnologías tuvo una mayor influencia en el diseño que su escalabilidad o rendimiento (el caso mas claro es el uso de Beans de Entidad CMP, dado que nuestro objetivo se centró en probar su flexibilidad y las ventajas que esta tecnología como parte de J2EE proporciona).

## **XDoclet**

Se utilizó XDoclet para la generación de los archivo de configuración, que si bien tienen una gran productividad luego de comprenderlos, no están debidamente documentados y en algunos casos es necesario incluir información para lograr los resultados buscados.

Uno de los problemas que encaran los desarrolladores J2EE es la sincronización de su código con los descriptores de despliegue J2EE. Según progresa el desarrollo de componentes, los desarrolladores tienen que ir actualizando los descriptores de despliegue, una tarea generalmente tediosa que puede provocar otros errores, en lugar de dedicar el tiempo a la lógica de negocio de la aplicación. XDoclet genera estas interfaces y clases de ayuda junto con los descriptores de despliegue, analizando los ficheros fuente. Estos ficheros se generan partiendo de plantillas que utilizan la información proporcionada en el código fuente y sus etiquetas JavaDoc. XDoclet permite a los

desarrolladores concentrarse en un único fichero fuente Java por cada componente, es decir, concentrarse en la lógica de negocio de la aplicación, y el resto lo hace XDoclet.

Durante muchos años los desarrolladores han estado buscando este tipo de herramientas que aceleran el proceso de desarrollo y despliegue, y XDoclet ciertamente es un paso en esa dirección.

XDoclet es un lenguaje basado en atributos. Permite reducir el tiempo de desarrollo generando automáticamente los archivos XML necesarios para realizar el despliegue, así como las clases asociadas a un Bean (interfaces locales y remotas, clases de claves compuestas, valores objetos, etc.). Por medio de esta tecnología se mantienen sincronizadas las diferentes interfaces de un Bean. Se caracteriza porque un Bean tiene asociada solamente una clase. También brinda un marco de trabajo independiente del servidor de aplicaciones para el que se esté desarrollando, lo que minimiza el impacto ante un cambio del mismo.

Esto es debido a que el mismo código XDoclet genera los diferentes archivos XML necesarios para configurar cada servidor de aplicaciones en particular.

El mecanismo utilizado para obtener el resultado anteriormente mencionado es la inclusión de metadata (por medio de etiquetas JavaDoc), tanto a nivel de clase como a nivel de método.

Para generar los diferentes archivos, XDoclet analiza tanto la metadata ingresada como la estructura del archivo fuente (paquetes, clases, métodos, y campos). Utilizando esta información y las diferentes plantillas para los archivos que se desean generar; se consigue como resultado un conjunto de clases java y de archivos XML necesarios para realizar el despliegue [18].

Esto es especialmente importante cuando se decide utilizar Beans de entidad CMP, pues XDoclet genera los mapeos específicos de cada CMP. Lamentablemente en este último punto no existe actualmente una



estandarización completa (es parcial) en las etiquetas a utilizar, por lo cual cada servidor de aplicaciones necesita que se le indique información extra mediante etiquetas propietarias [19].

## **SSL**

SSL (Secure Socket Layer) es una tecnología que permite a los navegadores y servidores Web a comunicarse bajo una conexión segura. En esa conexión, los datos que se envían son codificados antes de ser enviados y luego son descodificados para recibirlos y procesarlos. Ambos, el navegador y el servidor, codifican todo el tráfico antes de enviar cualquier dato. SSL mantiene las siguientes consideraciones de seguridad:

### **Autenticación.-**

Cuando se pretende establecer comunicación con un servidor Web sobre una conexión segura, éste presentará el navegador Web con un

conjunto de credenciales en forma de un certificado de servidor. El propósito del certificado es verificar que el sitio es el que dice ser. En algunos casos, el servidor puede pedir un certificado que indique que el cliente es quien dice ser (autenticación del cliente).

### **Confidencialidad.-**

Cuando los datos pasan a través del cliente y del servidor en una red, terceros pueden ver e interceptar estos datos. Las respuestas SSL son codificadas para que los datos no puedan ser descifrados por terceros y permanezcan seguros.

### **Integridad.-**

SSL ayuda a garantizar que los datos no sean modificados en el tránsito con terceros [3].

## **Tecnologías para la comunicación entre módulos**

Como en un sistema e-procurement, la comunicación entre usuarios y el sistema es vital, se utilizaron APIs de Java para la mensajería y correo electrónico. Estos son los siguientes:

JMS, utilizado para la interacción entre el módulo Storefront y la aplicación bancaria.

JavaMail, utilizado para envío de correo electrónico, es decir mantener la comunicación entre los usuarios y el sistema.

### **API para mensajería JMS**

JMS (Java Message Service) es un API de Java que permite a las aplicaciones crear, enviar, recibir y leer mensajes. JMS define un conjunto de interfaces en lenguaje Java, las cuales ayudan a la comunicación con otras implementaciones de mensajería [3]. Entre las características principales constan las siguientes:

Independencia de plataforma.

Confiabilidad: No permite repetición ni pérdida de mensajes.

Independiente del tiempo: Las implementaciones de proveedores JMS pueden enviar mensajes a medida que van llegando. Un cliente no necesita pedir mensajes, simplemente los recibe.

Desde la versión 1.3 de la plataforma J2EE, JMS puede ser usado para trabajar en conjunto con componentes J2EE.

### **API JavaMail**

JavaMail proporciona un API uniforme y común para manejar correo electrónico. Este permite a los proveedores del servicio suministrar una interfase estándar a sus sistemas usando lenguaje Java. Utilizando este API, se puede aplicar la tecnología mail en las aplicaciones Java y otorga la facilidad de acceder al almacén de mensajes, los compone y los envía [20].

Entre sus funciones están:

Crear mensajes.

Almacenar y recuperar mensajes.

Enviar mensajes.

Comunicación con el cliente, por ejemplo notificándole la  
llegada de un nuevo correo.

### 3.1.1 **Casos de uso**

Al módulo StoreFront de Eguana, durante el análisis, se lo puede dividir en secciones de funcionalidad. A continuación se listan las respectivas secciones:

Ingreso y salida del sistema

Catálogo

Carrito de compra

Proceso de compra

Proceso de despacho

Proceso de cancelación

Los casos de uso y sus respectivos escenarios son una parte fundamental del análisis de funcionalidad. A continuación se especifican los casos por sección de funcionalidad, además se especifican los escenarios más relevantes para cada caso de uso.<sup>1</sup>

---

<sup>1</sup> Orden, compra, y orden de compra se refieren al mismo concepto. Anular y cancelar también se refieren al mismo concepto. Aplicación bancaria y banco también se refieren al mismo concepto. Estas palabras son alternadas a lo largo del documento para evitar redundancias.

### **Ingreso y salida del sistema**

1. Ingresar usuario y clave.
  - 1.1. Usuario y clave correcto.
  - 1.2. Usuario y clave incorrecto.
  - 1.3. Uno o ambos campos vacíos.
2. Salir del sistema.
  - 2.1. Salir del sistema exitosamente.

### **Catálogo**

3. Buscar ítems de acuerdo a un criterio.
4. Buscar categorías de acuerdo a un criterio.
5. Buscar empresas de acuerdo a un criterio.
6. Ver ítems de acuerdo a la categoría.
7. Ver ítems de acuerdo a la empresa.

8. Ver detalles de un producto.

### **Carrito de compra**

9. Añadir un producto al carro de compra.

9.1. Añado un ítem al carro por primera vez en la sesión.

9.2. Añado un ítem de una empresa que no esta en el carro.

9.3. Añado un ítem de una empresa que ya esta en el carro.

9.4. Añado un ítem que ya esta en el carro.

10. Cambiar la cantidad de un ítem en el carro.

10.1. Poner una cantidad provista por el catálogo (si hay en stock).

10.2. Poner una cantidad no provista por el catálogo (no hay en stock).

10.3. Poner una cantidad en cero.



**Proceso de compra**

11. Ordenar la compra.

11.1. Compra exitosa, en proceso de espera para ser despachada y pagada.

12. Aprobar la compra.

12.1. Aprobación exitosa.

12.2. Aprobación fallida.

**Proceso de despacho**

13. Despachar la compra.

13.1. Compra despachada exitosamente.

**Proceso de anulación**

14. Usuario anula la orden.

14.1. Compra anulada exitosamente.

15. Aprobar la anulación de la orden.

15.1. Anulación aprobada.

15.2. Anulación no aprobada.

Para cada caso de uso y escenario arriba enumerado a continuación presentamos un análisis más detallado.

### Ingreso y salida del sistema

<b>Caso de uso 1</b>	<b>Ingresar usuario y clave</b>
Objetivo	Ingresar al sistema.
Precondiciones	Ninguna.
Resultados esperados	Usuario logra entrar al sistema de manera segura.
Actores	Cualquier usuario del sistema.
Descripción	El usuario ingresa su usuario y clave en la página de login del sistema.

### Escenarios:

<b>Escenario 1.1</b>	<b>Usuario y clave correcto</b>
Precondición	El usuario y clave ingresado coincide con el existente en la base de datos.
Resultado	El usuario ingresa al sistema Eguana.
Descripción	El usuario ingresa al sistema luego de ingresar exitosamente su usuario y clave.

<b>Escenario 1.2</b>	<b>Usuario y clave incorrecto</b>
Precondición	El usuario y clave ingresado no coincide con el existente en la base de datos.
Resultado	El usuario no ingresa al sistema Eguana, un mensaje de error apropiado es mostrado en el navegador.
Descripción	El usuario no puede ingresa al sistema luego de ingresar un usuario y clave erróneo.

<b>Escenario 1.3</b>	<b>Uno o ambos campos vacíos</b>
Precondición	Uno de los dos campos, usuario o clave, se encuentra vacío al intentar entrar al sistema.
Resultado	El usuario no ingresa al sistema Eguana, un mensaje de error apropiado es mostrado en el navegador.
Descripción	El usuario no puede ingresa al sistema luego de dejar uno de los campos, usuario o clave, vacío. No se permiten usuarios sin clave.

<b>Caso de uso 2</b>	<b>Salir del sistema</b>
Objetivo	Salir del sistema.
Precondiciones	El usuario se encuentra correctamente logoneado en el sistema Eguana.
Resultados esperados	Usuario logra salir del sistema, variables y demás recursos temporales son eliminados o reciclados.
Actores	Cualquier usuario del sistema.
Descripción	El usuario sale del sistema al hacer clic en el respectivo link en el navegador.

**Escenarios:**

<b>Escenario 2.1</b>	<b>Salir del sistema exitosamente</b>
Precondición	El usuario se encuentra correctamente logoneado en el sistema Eguana.
Resultado	Usuario logra salir del sistema, variables y demás recursos temporales son eliminados o reciclados.
Descripción	El usuario sale del sistema al hacer clic en el respectivo link en el navegador. Un mensaje de despedida aparece junto con un link para volver a ingresar si lo desea.

**Catálogo**

<b>Caso de uso 3</b>	<b>Buscar ítems de acuerdo a un criterio</b>
Objetivo	Ver todos los ítems en el catálogo que coincidan con el criterio de búsqueda especificado.
Precondiciones	Haber ingresado a la sección de compras.
Resultados esperados	Un listado en el navegador de todos los ítems que coincidan con el criterio de búsqueda especificado.
Actor	Usuario comprador.
Descripción	El usuario comprador ingresar un criterio de búsqueda en el campo respectivo, elige la opción "Descripción" y obtiene un listado de todos los ítems en el catálogo que coinciden con el criterio de búsqueda.

<b>Caso de uso 4</b>	<b>Buscar categorías de acuerdo a un</b>
----------------------	--

	<b>criterio</b>
Objetivo	Ver todas las categorías de productos en el catálogo que coincidan con el criterio de búsqueda especificado.
Precondiciones	Haber ingresado a la sección de compras.
Resultados esperados	Un listado en el navegador de todas las categorías de productos que coincidan con el criterio de búsqueda especificado.
Actor	Usuario comprador.
Descripción	El usuario comprador ingresa un criterio de búsqueda en el campo respectivo, elige la opción "Categoría" y obtiene un listado de todas las categorías en el catálogo que coinciden con el criterio de búsqueda.

<b>Caso de uso 5</b>	<b>Buscar empresas de acuerdo a un criterio</b>
Objetivo	Ver todas las empresas en el catálogo que coincidan con el criterio de búsqueda especificado.
Precondiciones	Haber ingresado a la sección de compras.
Resultados esperados	Un listado en el navegador de todas las empresas, que mantengan productos en el catálogo, que coincidan con el criterio de búsqueda especificado.
Actor	Usuario comprador.
Descripción	El usuario comprador ingresa un criterio de búsqueda en el campo respectivo, elige la opción "Empresa" y obtiene un listado de todas las empresas en el catálogo que coinciden con el criterio de búsqueda.

<b>Caso de uso 6</b>	<b>Ver ítems de acuerdo a la categoría</b>
Objetivo	Ver todos los ítems pertenecientes a una categoría de productos.
Precondiciones	Haber ingresado a la sección de compras.
Resultados esperados	Un listado en el navegador de todos los ítems pertenecientes a la categoría indicada.
Actor	Usuario comprador.
Descripción	El usuario hace clic en una categoría cualquiera y obtiene un listado de todos los productos en el catálogo, organizados por empresa, pertenecientes a esa categoría.

<b>Caso de uso 7</b>	<b>Ver ítems de acuerdo a la empresa</b>
Objetivo	Ver todos los ítems pertenecientes a una empresa.
Precondiciones	Haber ingresado a la sección de compras.
Resultados esperados	Un listado en el navegador de todos los ítems pertenecientes a la empresa indicada.
Actor	Usuario comprador.
Descripción	El usuario hace clic en una empresa cualquiera y obtiene un listado de todos los productos en el catálogo, organizados por categoría, pertenecientes a esa empresa.

<b>Caso de uso 8</b>	<b>Ver detalles de un producto</b>
Objetivo	Ver más información sobre un producto

	específico.
Precondiciones	Haber ingresado a la sección de compras.
Resultados esperados	Mostrar una página con todos los datos del producto.
Actor	Usuario comprador.
Descripción	El usuario hace clic en un producto y enseguida se muestra información detallada de este.

### Carrito de compra

<b>Caso de uso 9</b>	<b>Añadir un producto al carro de compra</b>
Objetivo	Agregar un producto más, el cuál se desea comprar, al carro de compras.
Precondiciones	Haber encontrado el respectivo producto en el catálogo.
Resultados esperados	El producto es añadido al carrito de compras el cuál es actualizado y mostrado al usuario.
Actor	Usuario comprador.
Descripción	El usuario hace clic en el link de “Añadir” de un producto y este respectivamente se añade al carro de compras el cuál aparece en pantalla con el precio, cantidad y total.

### Escenarios:

<b>Escenario 9.1</b>	<b>Añado un ítem al carro por primera vez en la sesión</b>
Precondición	El usuario nunca antes durante la sesión ha añadido un producto al carro de

	compras.
Resultado	Un nuevo carro de compras es creado en memoria. El resumen del carro de compras se empieza a mostrar en la sección derecha del navegador.
Descripción	El usuario añade su primer producto al carro de compras.

<b>Escenario 9.2</b>	<b>Añado un ítem de una empresa que no esta en el carro</b>
Precondición	El usuario nunca antes durante la sesión ha añadido un producto al carro de compras de esa respectiva empresa. El carro de compras no esta vacío.
Resultado	Se crea un nuevo objeto CompraEmpresa, se actualiza y muestra el carro de compras.
Descripción	El usuario añade su primer producto de una empresa a un carro de compras no vacío.

<b>Escenario 9.3</b>	<b>Añado un ítem de una empresa que ya esta en el carro</b>
Precondición	El usuario ha añadido anteriormente un producto al carro de compras de la respectiva empresa.
Resultado	Se actualiza y muestra el carro de compras.
Descripción	El usuario añade un producto más de una empresa a un carro de compras no vacío.

<b>Escenario 9.4</b>	<b>Añado un ítem que ya esta en el carro</b>
----------------------	--



Precondición	El usuario ha añadido anteriormente el mismo ítem al carro de compras.
Resultado	Se aumenta la cantidad en una unidad del respectivo ítem en el carro de compras.
Descripción	El usuario añade nuevamente el mismo ítem desde el catálogo y este aumenta su cantidad en una unidad en el carrito de compras el cuál es actualizado y mostrado.

<b>Caso de uso 10</b>	<b>Cambiar la cantidad de un ítem en el carro</b>
Objetivo	Cambiar la cantidad de un ítem mostrada en el carrito de compras.
Precondiciones	Tener ítems en el carro de compras.
Resultados esperados	La cantidad, total y resumen del carrito de compras es actualizada y mostrada.
Actor	Usuario comprador.
Descripción	El usuario cambia el campo de cantidad y hace clic en el botón de cambiar. El carrito de compras es actualizado.

### Escenarios:

<b>Escenario 10.1</b>	<b>Poner una cantidad provista por el catálogo (si hay en stock)</b>
Precondición	La nueva cantidad ingresada es menor o igual a la cantidad de ese ítem en stock en el catálogo.
Resultado	La cantidad es cambiada en el carro de compras. El carro es mostrado con la

	cantidad, subtotal y total actualizado.
Descripción	El usuario cambia exitosamente la cantidad del ítem en el carrito de compras.

<b>Escenario 10.2</b>	<b>Poner una cantidad no provista por el catálogo (no hay en stock)</b>
Precondición	La nueva cantidad ingresada es mayor a la cantidad de ese ítem en stock en el catálogo.
Resultado	Un mensaje de error apropiado es mostrado al usuario. El carrito de compras permanece igual.
Descripción	El usuario no logra cambiar la cantidad de un ítem en el carro de compras porque este sobrepasa la cantidad en stock.

<b>Escenario 10.3</b>	<b>Poner una cantidad en cero</b>
Precondición	La cantidad ingresada es cero.
Resultado	El ítem es eliminado del carrito de compras.
Descripción	El usuario ingresa el número cero en el campo cantidad, respectivamente el ítem es eliminado del carrito de compras y el total es actualizado.

### Proceso de compra

<b>Caso de uso 11</b>	<b>Ordenar la compra</b>
Objetivo	Comprar los ítems que existen en el carrito de compras.

Precondiciones	Tener ítems en el carro de compras.
Resultados esperados	Una o varias ordenes de compra es creada. Un email de confirmación es enviado al comprador y a los respectivos vendedores. Un mensaje asíncronico de la transacción es enviado a una aplicación bancaria externa.
Actor directo	Usuario comprador.
Actores indirectos	Usuarios vendedores del sistema, aplicación bancaria.
Descripción	El usuario hace clic en el botón "Ordenar" del carrito de compras. Se presente un resumen del carrito más los impuestos. Confirma nuevamente la acción y el carrito de compras se convierte en una o varias ordenes de compras. Un email de notificación se envía al usuario comprador y a los respectivos vendedores elegidos al azar para cada empresa en cada orden. El banco del comprador es notificado de la transacción y se espera su aprobación.

### Escenarios:

<b>Escenario 11.1</b>	<b>Compra exitosa, en proceso de espera para ser despachada y pagada</b>
Precondición	El usuario ha ordenado la compra.
Resultado	La orden de compra entra al estado "En Proceso".
Descripción	La orden de compra entra exitosamente al estado "En Proceso" luego de haber sido creada.

<b>Caso de uso 12</b>	<b>Aprobar la compra</b>
Objetivo	La aplicación debe de aprobar la transacción de la compra.
Precondiciones	Haber creado exitosamente una orden de compra.
Resultados esperados	La transacción de la orden de compra es aprobada por el banco del usuario comprador. Se hace un debito a la cuenta del usuario comprador y acreditaciones a las cuentas de las respectivas empresas vendedoras.
Actor directo	Aplicación bancaria.
Actores indirectos	Usuario comprador y vendedor.
Descripción	Una vez que la orden es creada se envía un mensaje asíncronico a la aplicación del banco del comprador. Esta responde con un mensaje de aprobación o anulación de la transacción poniendo la compra en estado "Pagada" o "Anulada" respectivamente. Un email de información se envía al comprador y vendedor.

### Escenarios:

<b>Escenario 12.1</b>	<b>Aprobación exitosa</b>
Precondición	El usuario ha ordenado la compra y el total de ésta no excede el saldo de la cuenta bancaria de su empresa o la compra no contradice las políticas del banco.
Resultado	La orden de compra entra al estado "Pagada". Se envían emails de confirmación al comprador y vendedor.
Descripción	La orden de compra entre exitosamente

	al estado "Pagada" luego de haber sido aprobada por el banco. Un email de información se envía al comprador y vendedor.
--	---

<b>Escenario 12.2</b>	<b>Aprobación fallida</b>
Precondición	El usuario ha ordenado la compra y el total de ésta excede el saldo de la cuenta bancaria de su empresa o la compra contradice las políticas del banco.
Resultado	La orden de compra entra al estado "No aprobada". Se envían emails de confirmación al comprador y vendedor.
Descripción	La orden de compra entra al estado "No aprobada" luego de haber sido negada la transacción por el banco. Un email de información se envía al comprador y vendedor.

### Proceso de despacho

<b>Caso de uso 13</b>	<b>Despachar la compra</b>
Objetivo	Despachar los productos comprados por el usuario comprador.
Precondiciones	La orden de compras debe estar en estado pagada.
Resultados esperados	La orden de compra entra al estado "despachada" y los ítems son enviados al usuario comprador a través de cualquier medio de transportación que prefiera la empresa. Se envían un email de confirmación al comprador.
Actor directo	Usuario vendedor.

Actores indirectos	Usuario comprador.
Descripción	El usuario vendedor entra a la sección de órdenes y marca una orden de compra en estado pagada como despachada. Un email se envía al comprador notificando el despacho de la orden.

### Escenarios:

<b>Escenario 13.1</b>	<b>Compra despachada exitosamente</b>
Precondición	La orden de compra debe de estar en el estado "Pagada".
Resultado	La orden de compra entra al estado "Despechada". Los productos son enviados al usuario comprador por la empresa vendedora. La transacción finaliza.
Descripción	El vendedor al entrar a la interfase de ordenes del módulo StoreFront revisa que la orden de compra asignada esta en el estado "Pagada". A continuación procede a marcarla como despachada finalizando la transacción.

### Proceso de anulación

<b>Caso de uso 14</b>	<b>Usuario anula la orden</b>
Objetivo	Anular la orden hecha por el comprador.
Precondiciones	Existe una orden de compra en estado "En Proceso" o "Pagada".
Resultados esperados	La orden de compra entra al estado

	“Anulada” y el dinero es devuelto a la cuenta de la empresa del comprador. Se envían un email de confirmación al comprador y al vendedor.
Actor directo	Usuario comprador o vendedor.
Actores indirectos	Usuario vendedor o comprador.
Descripción	El usuario comprador (o vendedor) entra a la sección de órdenes y marca una orden como anulada. El usuario espera que la transacción monetaria sea revertida.

### Escenarios:

<b>Escenario 14.1</b>	<b>Compra anulada exitosamente</b>
Precondición	La aplicación bancaria ha recibido un mensaje de anulación de una compra en estado “En Proceso” o “Pagada”.
Resultado	La orden de compra espera en el estado “En proceso de Anulación”.
Descripción	La orden de compra al ser anulada por el usuario entra inmediatamente al estado “En proceso de Anulación” y el sistema espera el mensaje de respuesta de la aplicación bancaria confirmando la anulación de la transacción monetaria.

<b>Caso de uso 15</b>	<b>Aprobar la anulación de la orden</b>
Objetivo	La aplicación bancaria debe de aprobar la anulación requerida por un usuario de Eguana.
Precondiciones	Existe una orden de compra en estado “En Proceso de Anulación”.
Resultados esperados	La orden de compra entra al estado

	"Anulada" y el dinero es devuelto a la cuenta de la empresa del comprador.
Actor directo	Aplicación bancaria.
Descripción	La aplicación bancaria al recibir un mensaje asincrónico de Eguana pidiendo la anulación de una transacción procede a enviar un mensaje de respuesta al sistema informando el éxito o falla de la anulación.

### Escenarios:

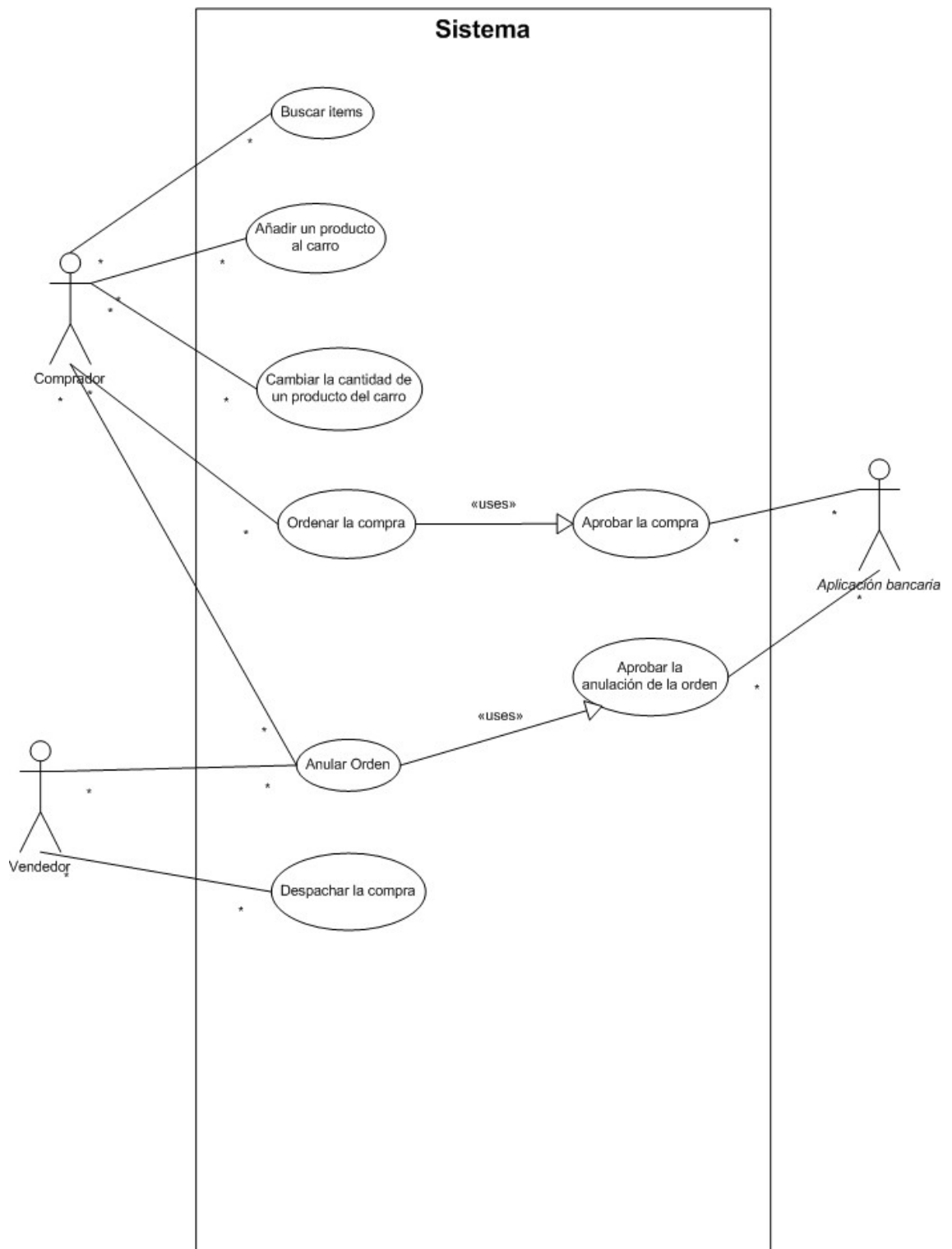
<b>Escenario 15.1</b>	<b>Anulación aprobada</b>
Precondición	La aplicación bancaria ha recibido un mensaje de anulación de una compra en estado "En Proceso" o "Pagada".
Resultado	La orden de compra termina en el estado "Anulada". Se envían emails de confirmación al usuario comprador y vendedor.
Descripción	La orden de compra al ser anulada por el usuario entra inmediatamente al estado "En proceso de Anulación" y el sistema espera el mensaje de respuesta de la aplicación bancaria confirmando la anulación de la transacción monetaria. Una vez confirmada la anulación la orden de compra entra al estado "Anulada".

<b>Escenario 15.2</b>	<b>Anulación no aprobada</b>
Precondición	La aplicación bancaria ha recibido un mensaje de anulación de una compra en estado "En Proceso" o "Pagada". Algún problema dentro de la institución



	bancaria no permita la anulación de la transacción monetaria.
Resultado	La orden de compra termina en el estado inicial "En Proceso" o "Pagada". Se envían emails de confirmación al usuario comprador y vendedor.
Descripción	La orden de compra al ser anulada por el usuario entra inmediatamente al estado "En proceso de Anulación" y el sistema espera el mensaje de respuesta de la aplicación bancaria confirmando la anulación de la transacción monetaria. En el caso que la aplicación bancaria responda de manera negativa a la solicitud la orden de compra volverá a su estado anterior a la solicitud de anulación.

A continuación se presenta un diagrama de los casos de uso más importantes en el proyecto:



**Figura 10: Casos de uso**

### 3.1.2 Diagramas de estado

Los diagramas de estado nos muestran el cambio de estado de los objetos en el sistema a través del tiempo. Estos diagramas son fundamentales en el análisis dinámico de un sistema porque nos proporcionan una vista en la dimensión temporal del sistema.

En Eguana existen muchos objetos cuyo estado cambia a través del tiempo como por ejemplo: Usuario, Empresa, Compra, Rol, etc. La mayoría de estos objetos no son manejados directamente por el módulo StoreFront, por lo tanto el estado de éstos no entra en el análisis de este módulo. La excepción es el objeto Compra, este objeto es el corazón del módulo StoreFront. En este modulo todo gira alrededor del objeto Compra haciendo que su diagrama de estado sea de vital importancia en el análisis.

# Diagrama de estado del objeto Compra

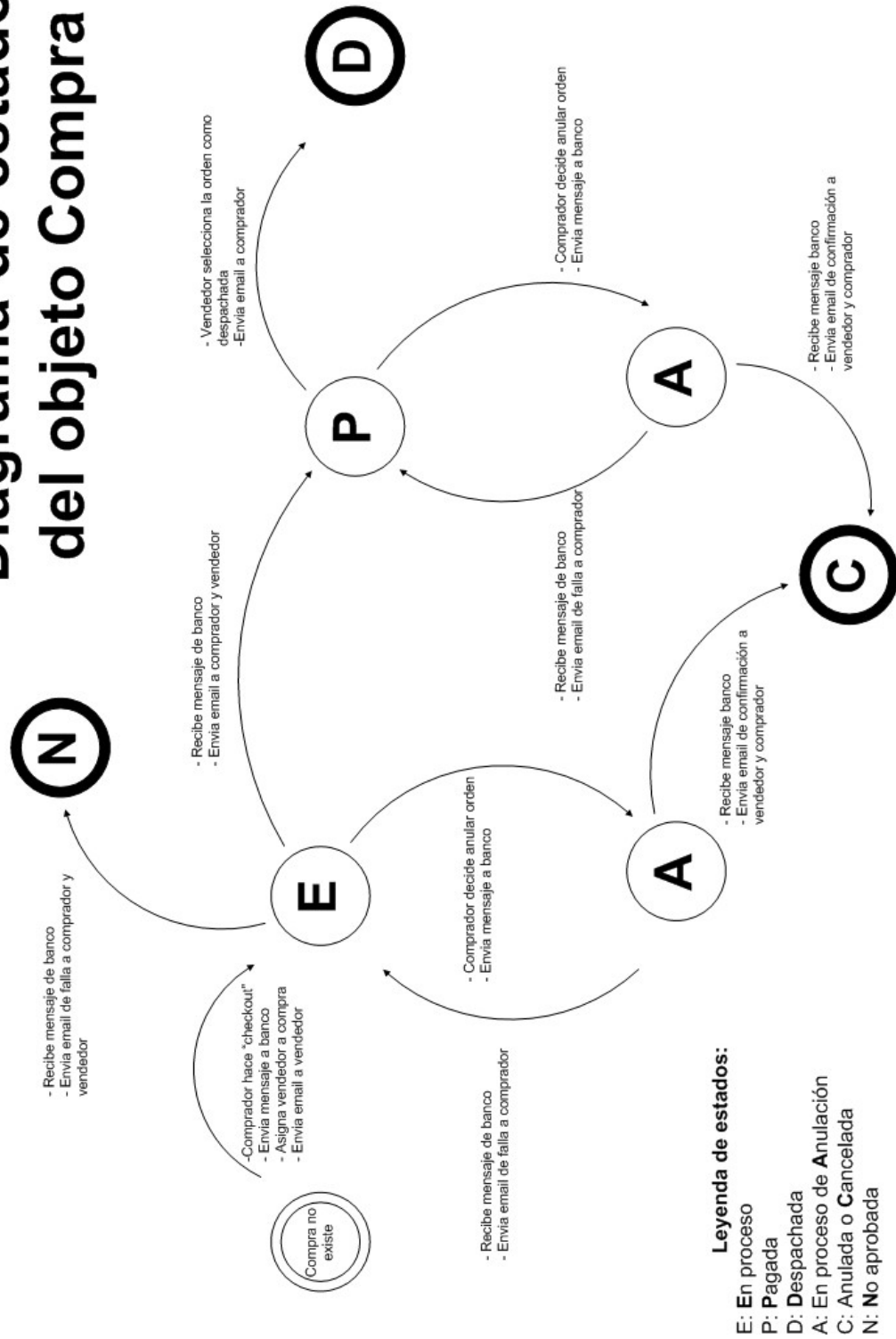


Figura 11: Diagrama de estado del objeto Compra

En la Figura 11 encontramos el diagrama de estado del objeto “Compra” u “Orden” como también es referido. En la sección 3.1.1 se discutió mucho en los casos de uso sobre estos estados. A continuación sigue una explicación detallada de este diagrama y sus respectivos estados:

Primero se obtiene el estado inicial en donde la compra no existe. Esto es cuando la compra todavía es representada como un carrito de compras. Una vez que el usuario comprador hace clic en el botón ordenar, el carrito se divide en una orden de compra por cada empresa que esté presente en el carrito.

Cuando una orden de compra es creada a partir del carrito de compras, ésta entra inmediatamente al estado “En Proceso” (E). En este estado, el sistema asigna un vendedor de la empresa vendedora escogido al azar. Se le envía un email al vendedor. Luego se notifica a la aplicación bancaria del banco de la empresa del comprador que apruebe la compra, debitando el total de su cuenta y acreditándolo a la cuenta en el

banco de la empresa vendedora por medio de una transferencia bancaria. En el caso que el banco niegue la aprobación por cualquier motivo la compra pasara al estado “No Aprobada” (N). Si el usuario comprador se arrepiente y decide cancelar la compra esta entra al estado “En proceso de Anulación”. Inmediatamente se envía un mensaje a la aplicación bancaria para que reverse la transacción y un email al comprador y al vendedor informando de la cancelación. Si el banco logra revertir la transacción exitosamente la compra entra al estado “Anulada o Cancelada” (C) de lo contrario vuelve al estado (E). La comunicación con la aplicación bancaria se la hace a través de mensajes asincrónicos.

Una vez que el banco envíe un mensaje de pago exitoso la compra entra al estado “Pagada” (P). En este estado aún puede ser cancelada por el comprador e incluso por el vendedor y la orden seguiría el mismo proceso de anulación del estado (E). El vendedor tiene la potestad de cancelar una compra en el caso que ya no tenga el producto en stock o

su empresa no pueda garantizar la entrega del producto por cualquier razón. En el estado (P) el vendedor además tiene la opción de despachar la orden desde el sistema. Una vez que el vendedor marque a la compra como despachada esta entra automáticamente a su estado final “Despachada” (D). La orden no puede ser cancelada en este estado.

### 3.1.3 Diagramas de interacción de objetos

Los diagramas de interacción de objetos (DIO) como su mismo nombre lo sugiere muestran la interacción entre los objetos del sistema para cada escenario especificado en los casos de uso. Estos diagramas son parte del análisis dinámico del sistema y especifican de una manera más detallada la funcionalidad interna de los componentes de

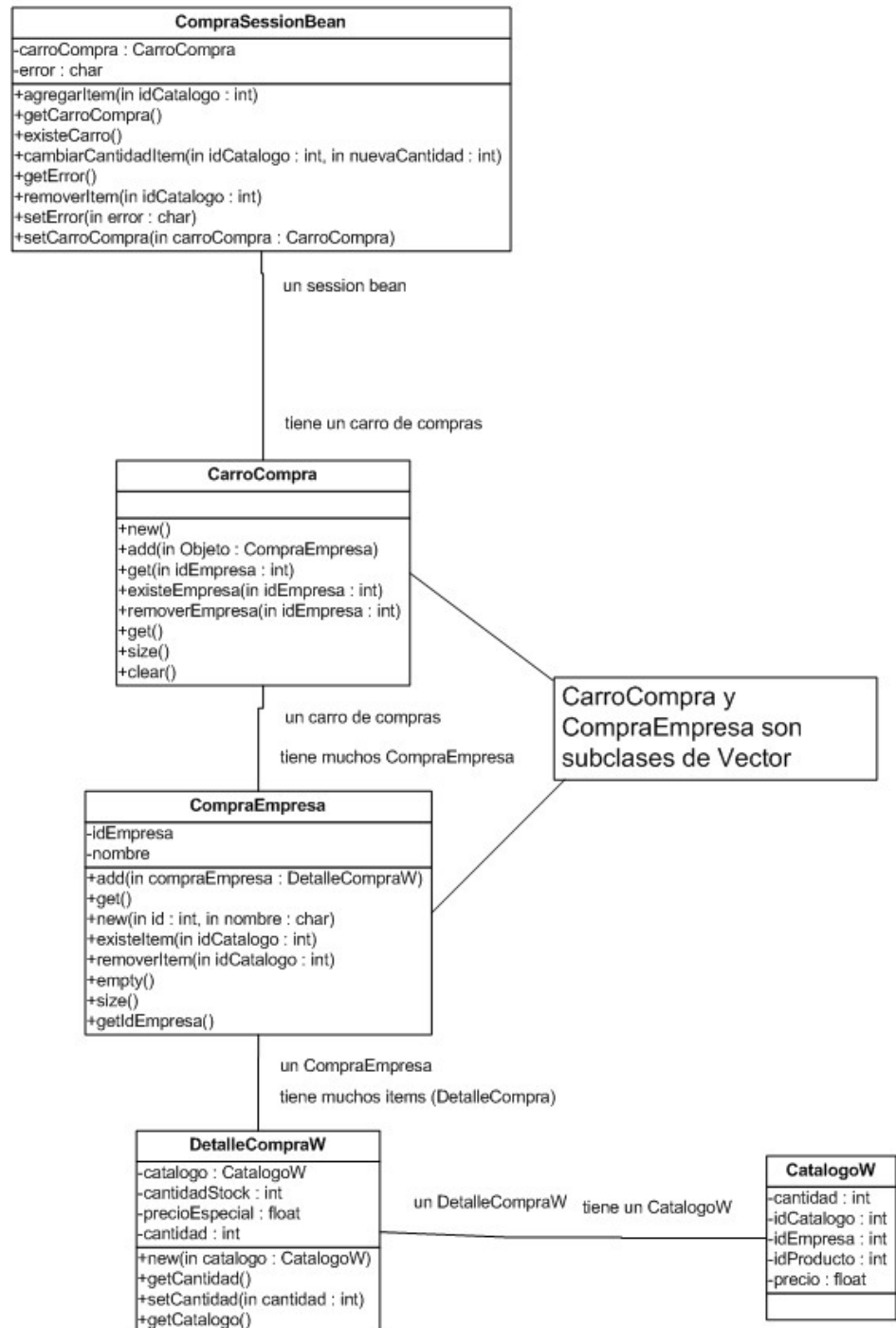
StoreFront. Hacer estos diagramas es un paso obligatorio antes del diseño y la programación. Por razones de brevedad no se mostraran los diagramas para cada uno de los escenarios del sistema. Además en cada escenario se mostraran solamente los objetos más importantes. A continuación mostramos los diagramas más relevantes:

### **Carrito de compras**

Para poder comprender mejor los DIOs se incluye además un diagrama estático de los principales objetos involucrados en el carro de compras. Solo se incluyen en este diagrama los atributos y métodos usados en los DIOs.



## Diagrama de objetos para el carro de compras



**Figura 12: Diagrama estático de objetos**

Escenario 9.1: Añade un ítem por primera vez al carro

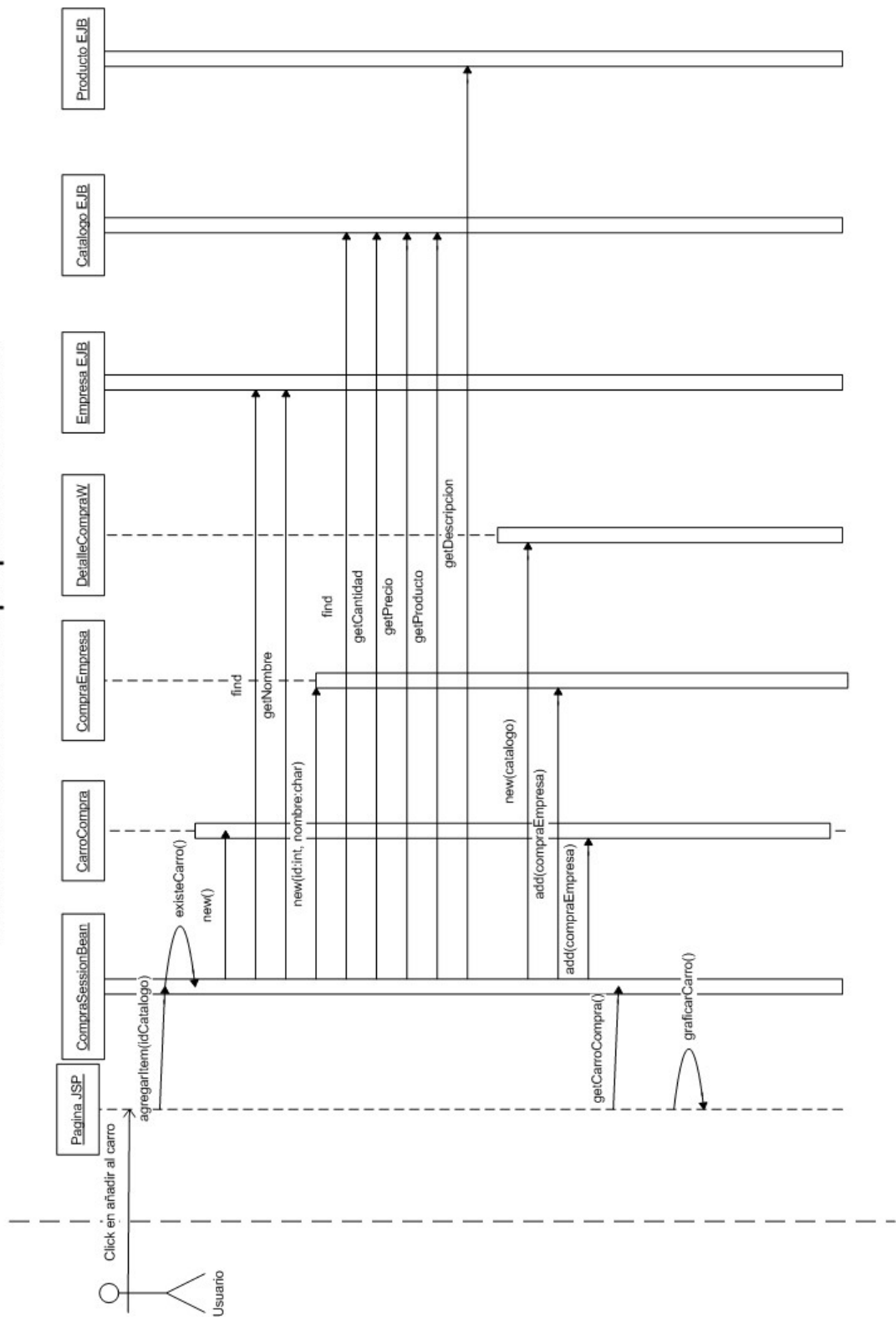


Figura 13: Escenario 9.1

### Escenario 9.4: Añade un item que ya esta en el carro

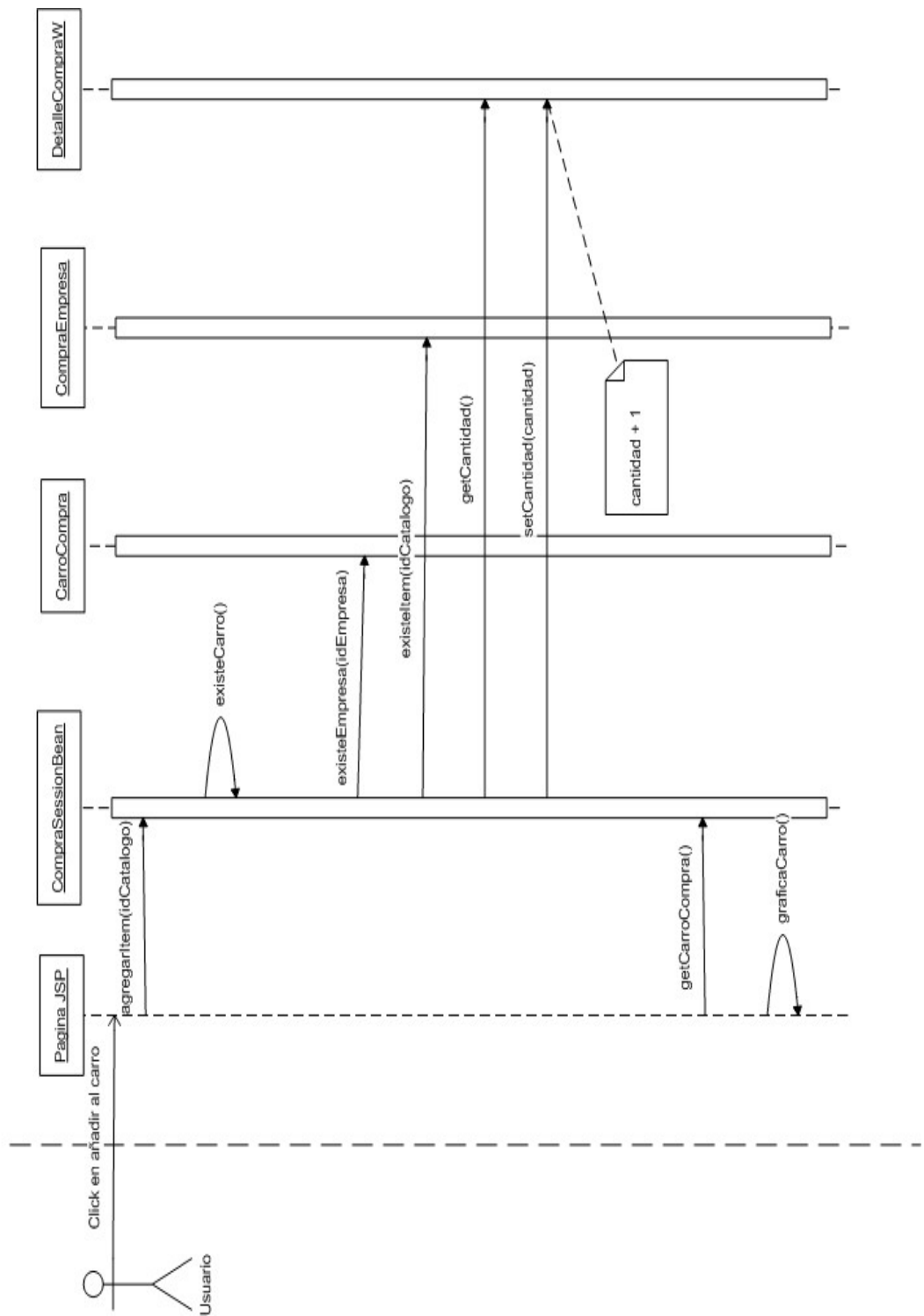
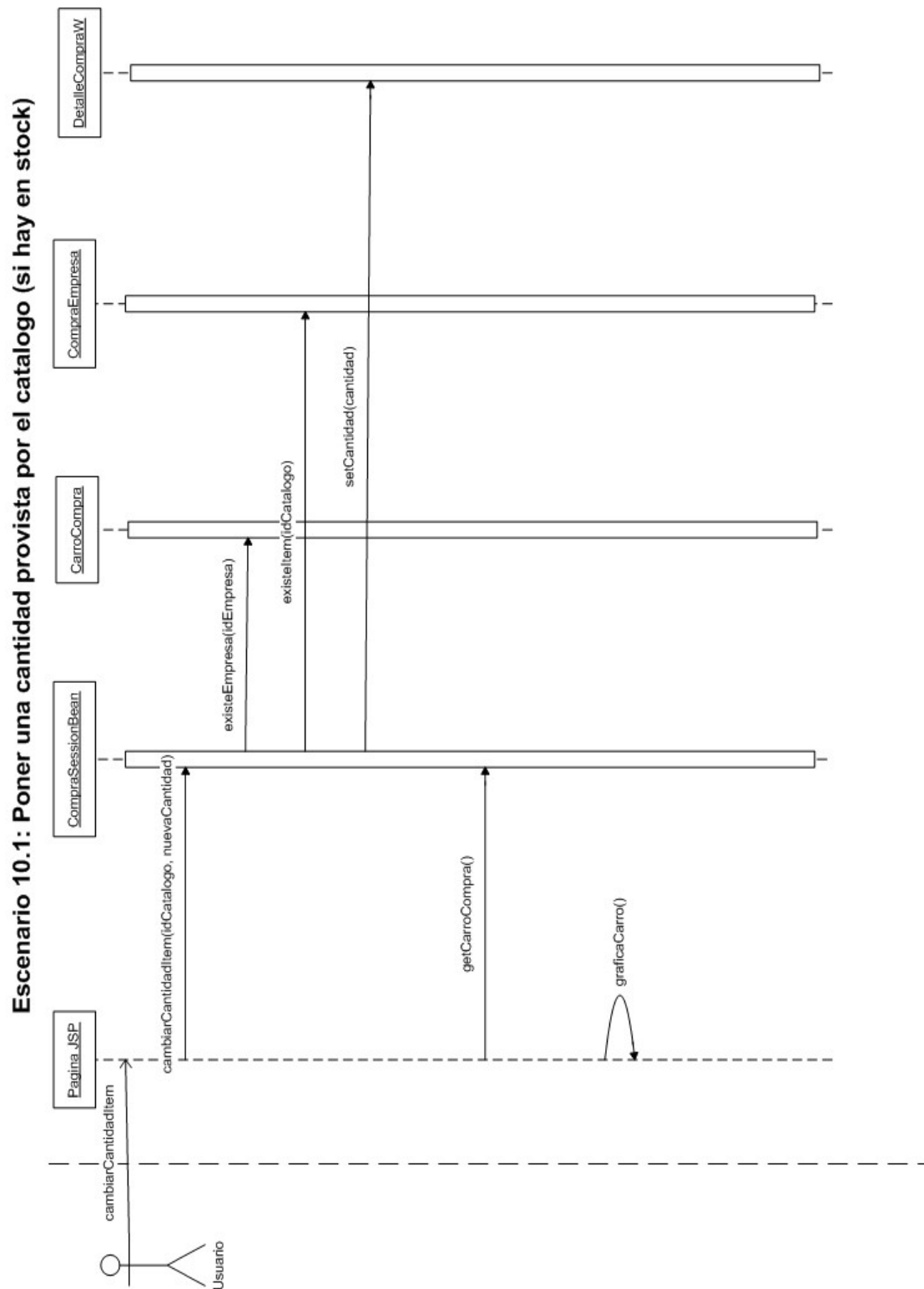
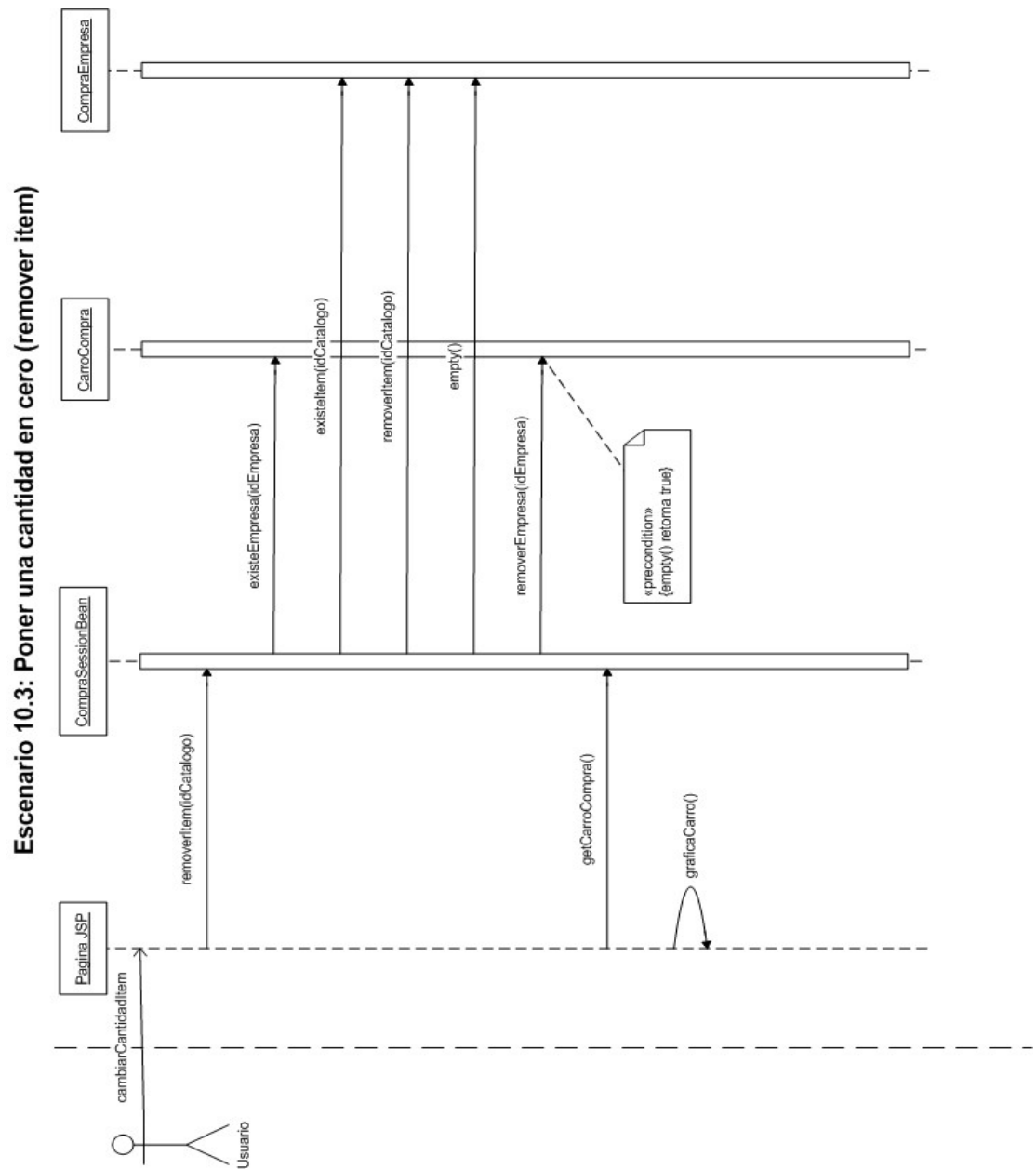


Figura 14: Escenario 9.4



**Figura 15: Escenario 10.1**

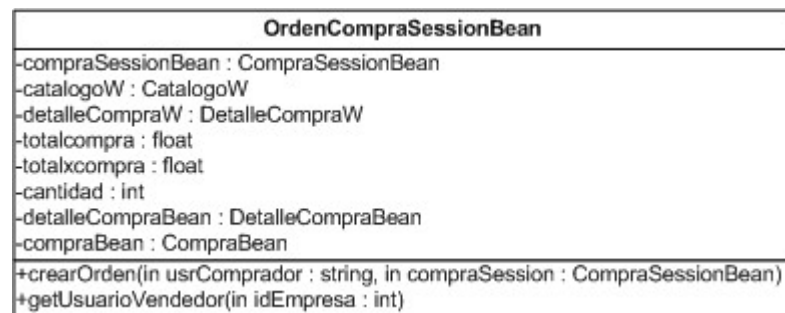


**Figura 16: Escenario 10.3**

En los escenarios arriba mostrados se asume que el actor del sistema es siempre un usuario comprador ya que solamente este puede navegar en el catálogo y añadir productos al carrito de compras.

## Proceso de compras, despacho y anulación

Para poder comprender mejor los DIOs se incluye un diagrama estático de OrdenCompraSessionBean. Solo se incluyen en este diagrama los atributos y métodos usados en los DIOs.



**Figura 17: Diagrama estático de OrdenCompraSessionBean**

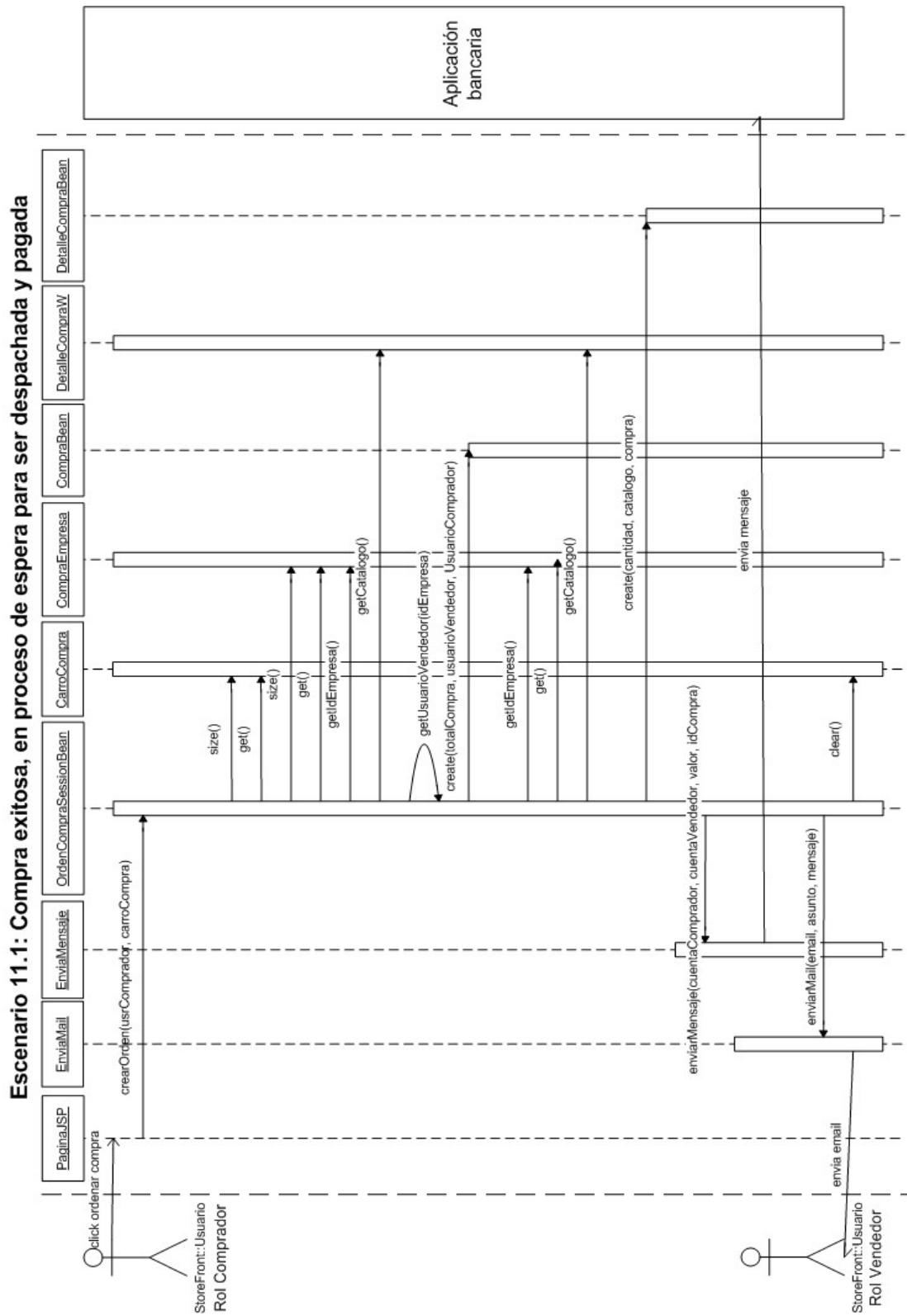
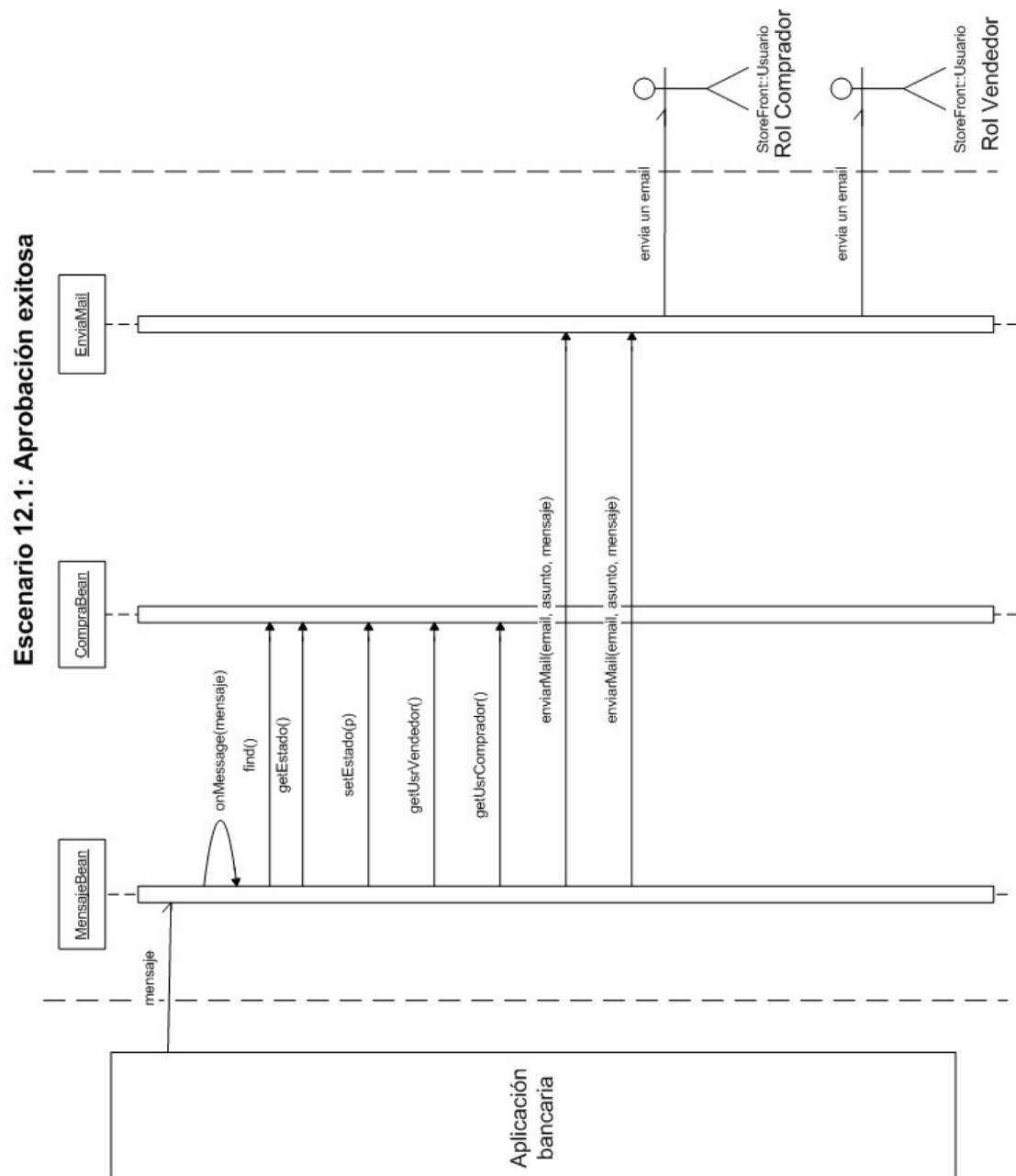


Figura 18: Escenario 11.1

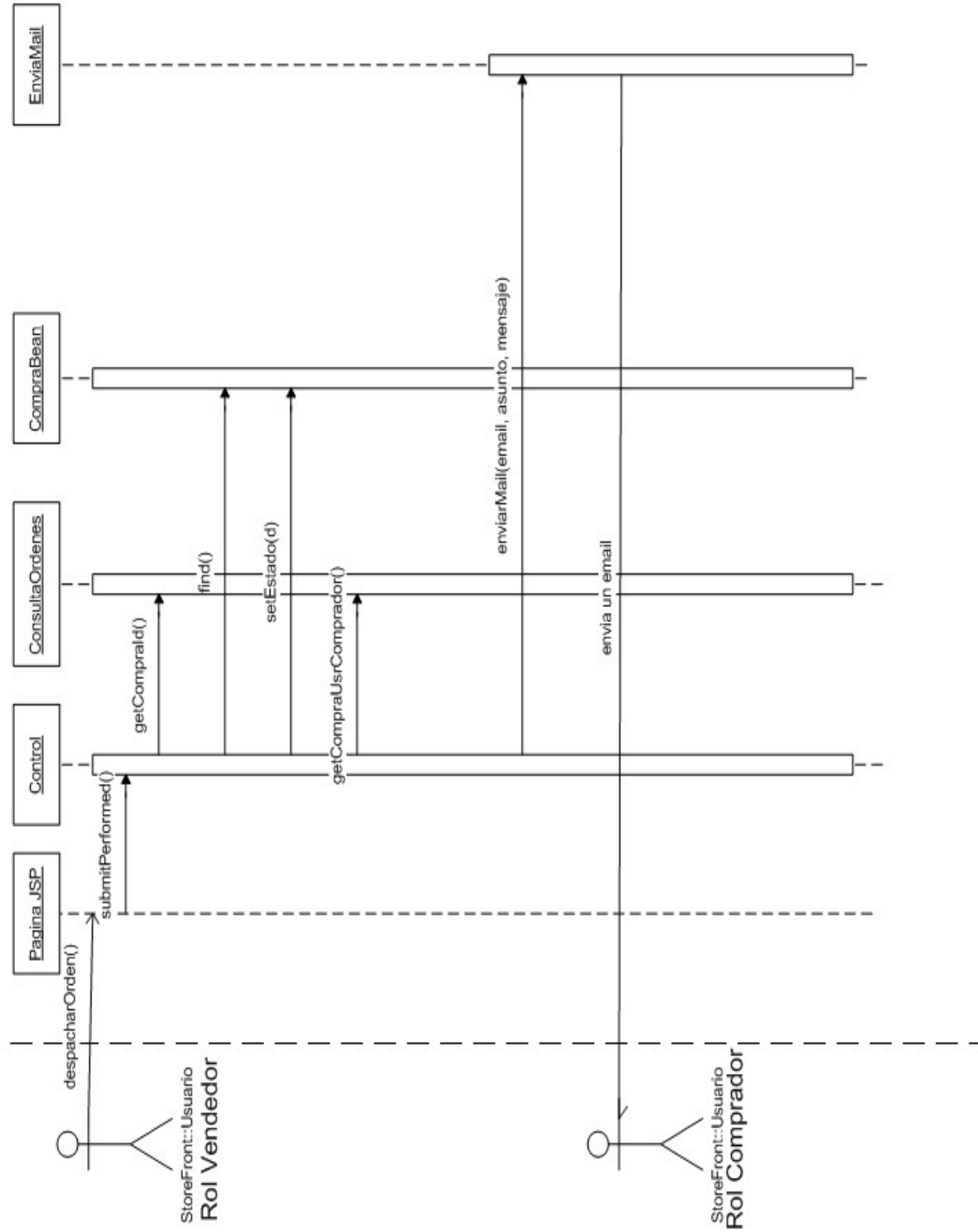


En el DIO del escenario 11.1 se asume que el carro de compras contiene un solo ítem, en el caso contrario simplemente se iteran algunas funciones del diagrama.



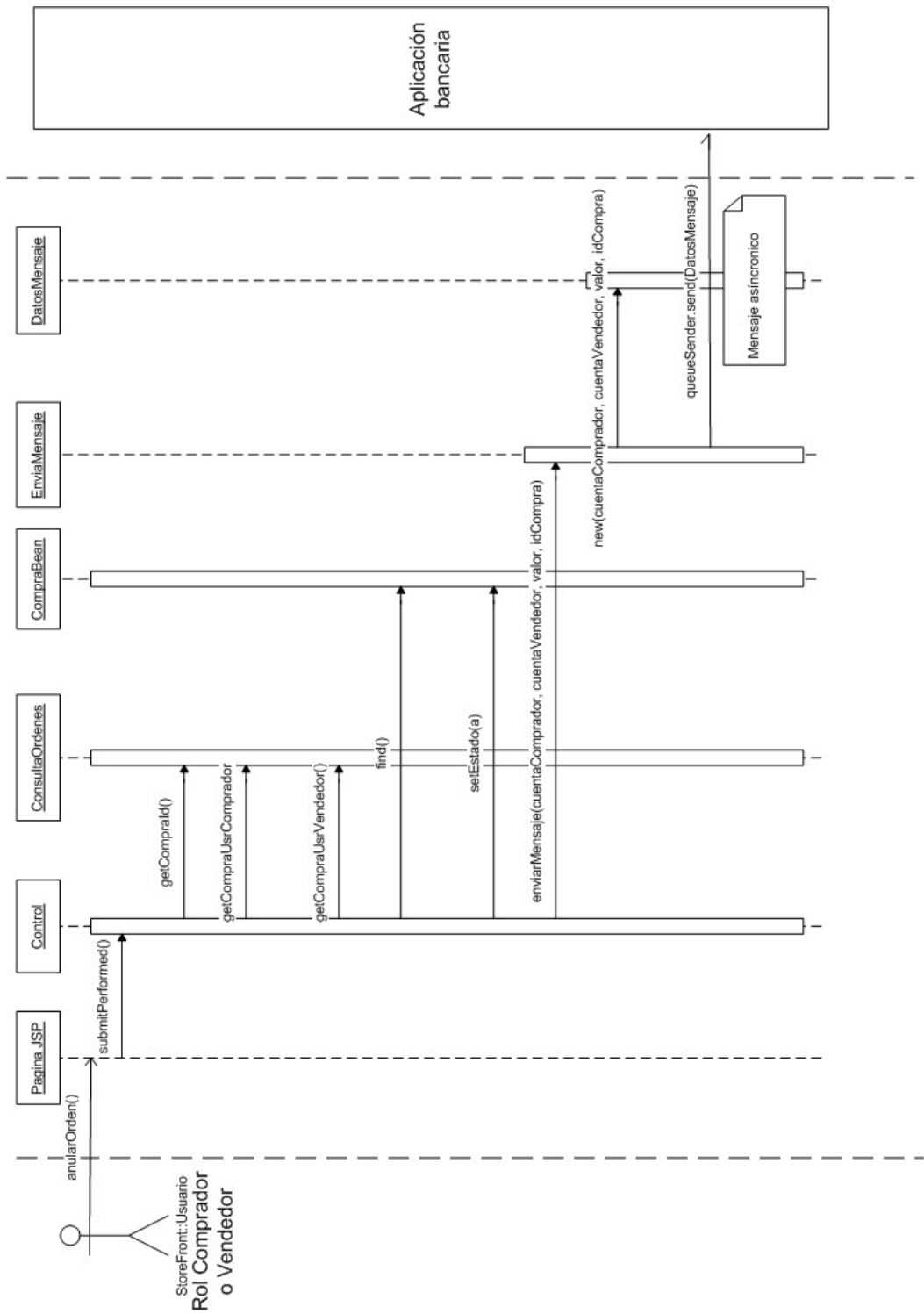
**Figura 19: Escenario 12.1**

**Escenario 13.1: Compra despachada exitosamente**



**Figura 20: Escenario 13.1**

**Escenario 14.1: Compra anulada exitosamente**



**Figura 21: Escenario 14.1**

La aplicación bancaria por ser externa al sistema es considerada como un actor del sistema. Algunos Beans y EJBs no se han mostrado para poder hacer más entendibles los diagramas.

Los diagramas presentados documentan de manera general el comportamiento de los objetos del módulo StoreFront. En la etapa de diseño e implementación estos objetos son pulidos con más detalle para acoger la funcionalidad requerida por J2EE y JBOSS.

#### 3.1.4 Ingreso al sistema

##### ***Antecedentes***

Existen dos aspectos importantes que se deben considerar para asegurar aplicaciones Web o J2EE: [21]

Evitar el acceso a datos confidenciales a usuarios no autorizados.

Prevenir el robo de datos de la red mientras los datos estén en tránsito.

Siendo Eguana un sistema cuyo ambiente es el Web, estas dos consideraciones de seguridad son vitales para su funcionamiento, por tanto fueron tomadas en consideración dentro del esquema de seguridades de dos maneras:

A través del uso de procesos de autenticación y autorización.

A través del uso de tecnología SSL, que permite encriptar el tráfico entre el navegador y el servidor.

### **Autenticación y autorización**

Las aplicaciones J2EE consisten de componentes que pueden contener recursos protegidos y no protegidos. A menudo, se necesita proteger recursos que aseguren que solamente usuarios autorizados tengan acceso. La autorización provee acceso controlado a recursos protegidos y se basa en la identificación y en la autenticación. La identificación es un proceso que permite a un sistema reconocer a una entidad y la autenticación verifica la identidad de un usuario, dispositivo u otra entidad en un sistema computacional, usualmente como un prerrequisito para permitir el acceso a recursos en un sistema.

La autenticación se basa normalmente en un usuario y contraseña. En los sistemas de seguridad, la autenticación es distinta de la autorización porque la autenticación asegura solamente que el usuario es quien dice ser, pero no menciona nada sobre sus derechos de acceso al sistema, de lo cual se encarga el proceso de autorización.

Cuando se trata de dar acceso a un recurso Web protegido, el contenedor Web activa el mecanismo de autenticación que ha sido configurado para este recurso. De no especificarse algún mecanismo, el usuario no será autenticado.

Para el módulo Storefront, el tipo de autenticación seleccionado fue la autenticación basada en formulario. Este tipo de autenticación usa una forma HTML normal para ingresar la información de usuario y contraseña. Esta información es enviada al servidor, el cual maneja la autenticación una sola vez.

La autenticación basada en un formulario es manejada completamente por el contenedor servlet. Lo único que tiene que hacer el desarrollador es especificar en el archivo Web.xml cuáles páginas (URLs) son seguras. La figura a continuación describe gráficamente este proceso:

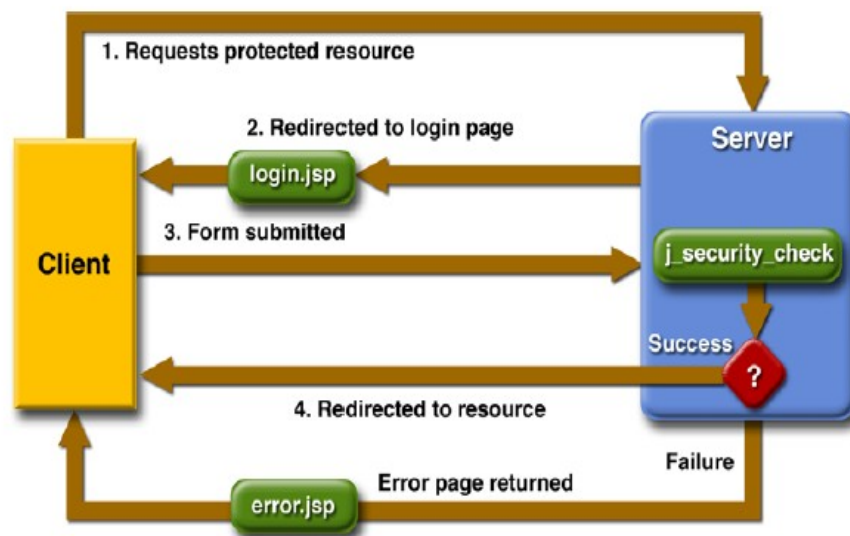


Figura 22: Autenticación basada en un formulario [3]

Como podemos observar, se suscitan los siguientes pasos:



1. Un requerimiento del cliente pide acceso a un recurso protegido.
2. Si el cliente no es autenticado, el servidor redirecciona el cliente a la página de inicio de sesión (login).
3. El cliente envía el formulario de inicio de sesión al servidor.
4. Si el inicio de sesión es exitoso, el servidor redirecciona el cliente al recurso. Si falla, el cliente es redireccionado a una página de error.

Esta forma de autenticación permite personalizar las pantallas de login y de errores que son presentadas al usuario final en un navegador, pero no es particularmente segura porque puede exponer la información del usuario debido a que es enviada como un archivo plano y puede ser fácilmente decodificada, a menos que todas las conexiones estén sobre SSL. Por esto, Storefront utilizará también la tecnología SSL para complementar los niveles de seguridad de la aplicación porque brinda la

facilidad de que no cambia la manera de trabajar de la autenticación basada en un formulario ni altera servlets o páginas JSP. Todo es configurable a nivel de descriptores.

### **Seguridades en los contenedores Web**

A nivel de la aplicación Web en sí, como éstas están compuestas de componentes que pueden ser desplegados en diferentes contenedores, los cuales componen toda la aplicación, las seguridades son provistas por estos últimos.

Un contenedor provee dos clases de seguridades: seguridad declarativa y programática.

#### **Seguridad Declarativa.-**

Expresa una estructura de seguridad de una aplicación, que incluye roles, control de acceso y requerimientos de autenticación, en una forma externa a la aplicación (en un descriptor) [3].

### **Seguridad Programática.-**

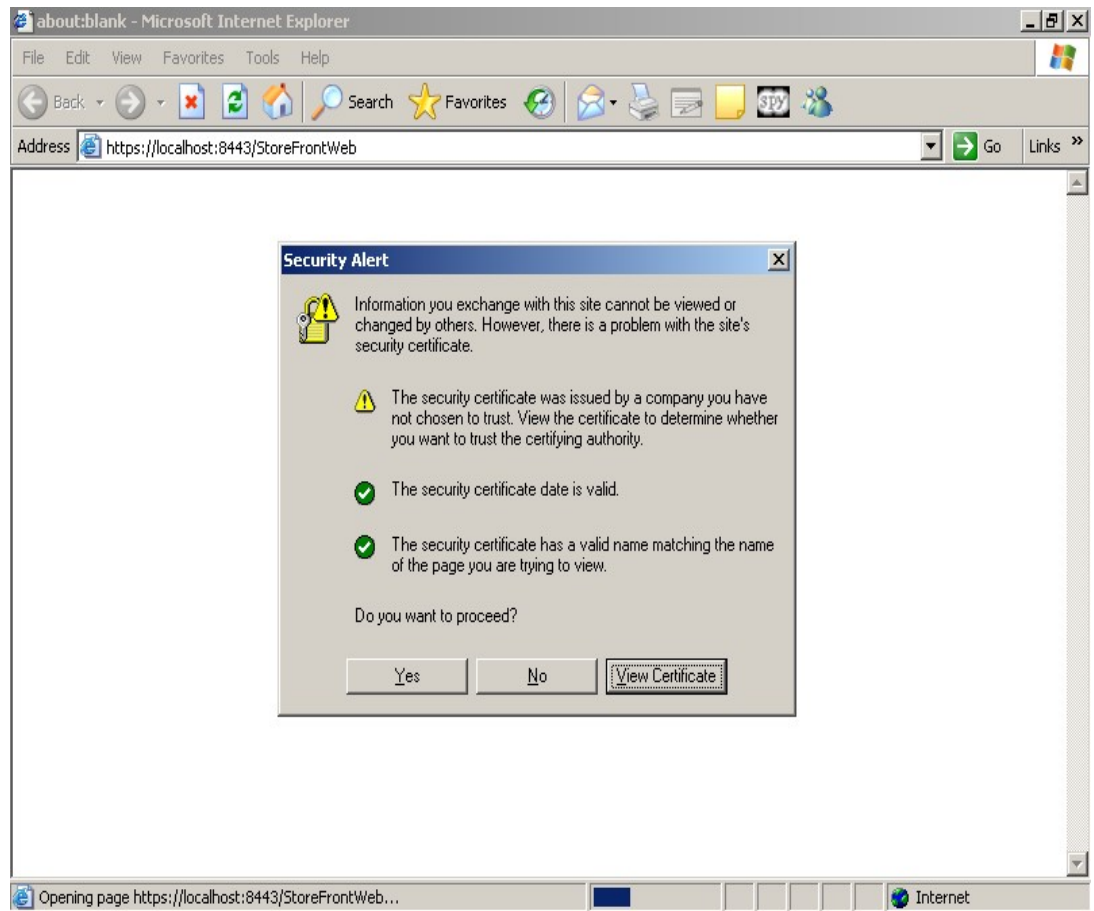
Este tipo de seguridad es embebida en la aplicación y es usada para crear decisiones de seguridad. Es útil cuando la seguridad declarativa sola no es suficiente para expresar el modelo de seguridad de la aplicación [3].

Debido a la personalización de páginas jsp y a la facilidad y transparencia que brinda el servidor JBoss para configuraciones en descriptores, el módulo Storefront de Eguana utilizará los dos tipos de seguridades para aprovechar todas las ventajas que éstos presentan para armar adecuadamente todo el esquema de seguridad que se desea para el sistema.

***Funcionalidad***

De acuerdo a las especificaciones de seguridades expuestas y para mantener el sistema protegido de la intervención de terceros, se realizó la configuración en el servidor JBoss para que soporte SSL y manejar la aplicación sobre una conexión segura. Así, para levantar el sistema el usuario se conectará al mismo en un navegador Web a través del protocolo HTTPS.

La primera vez que ingrese al sitio, el navegador mostrará una pantalla de alerta indicando el nivel de seguridad al que está expuesta la aplicación y que está utilizando un certificado de prueba, por lo que se ha creado un conflicto. El usuario debe escoger la opción de proceder la ejecución hasta que se autorice el certificado.



**Figura 23: Alerta previa a página de inicio de Eguana**


A continuación se presentará la página de inicio del sistema, donde se encuentra el formulario de inicio de sesión en el cual el usuario ingresa sus datos de usuario y contraseña. El usuario debe haber sido registrado previamente por su empresa en el sistema y, dependiendo de su tarea se le habrá asignado un usuario, clave y roles.

E-guana .: E-procurement Open Source.: - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Refresh Print Mail Stop Spy

Address <https://localhost:8443/StoreFrontWeb/paginas/index.jsp> Go Links >>



**e-guana** E-PROCUREMENT OPEN SOURCE

### Bienvenido a Eguana E-Procurement

Ingreso a Eguana:

ESTE ES EL PRIMER MERCADO VIRTUAL ENTRE EMPRESAS DEL ECUADOR.

REALIZADO POR ESTUDIANTES DE LA ESPOL, FUE DISEÑADO PARA QUE LAS DIFERENTES EMPRESAS DEL PAIS PUEDAN REALIZAR DIVERSAS ACTIVIDADES COMERCIALES EN INTERNET CON SOLO REGISTRARSE A NUESTRO SISTEMA.

Usuario:

Clave:

ALL CONTENT COPYRIGHT ©2004 EGUANA

Done Local intranet

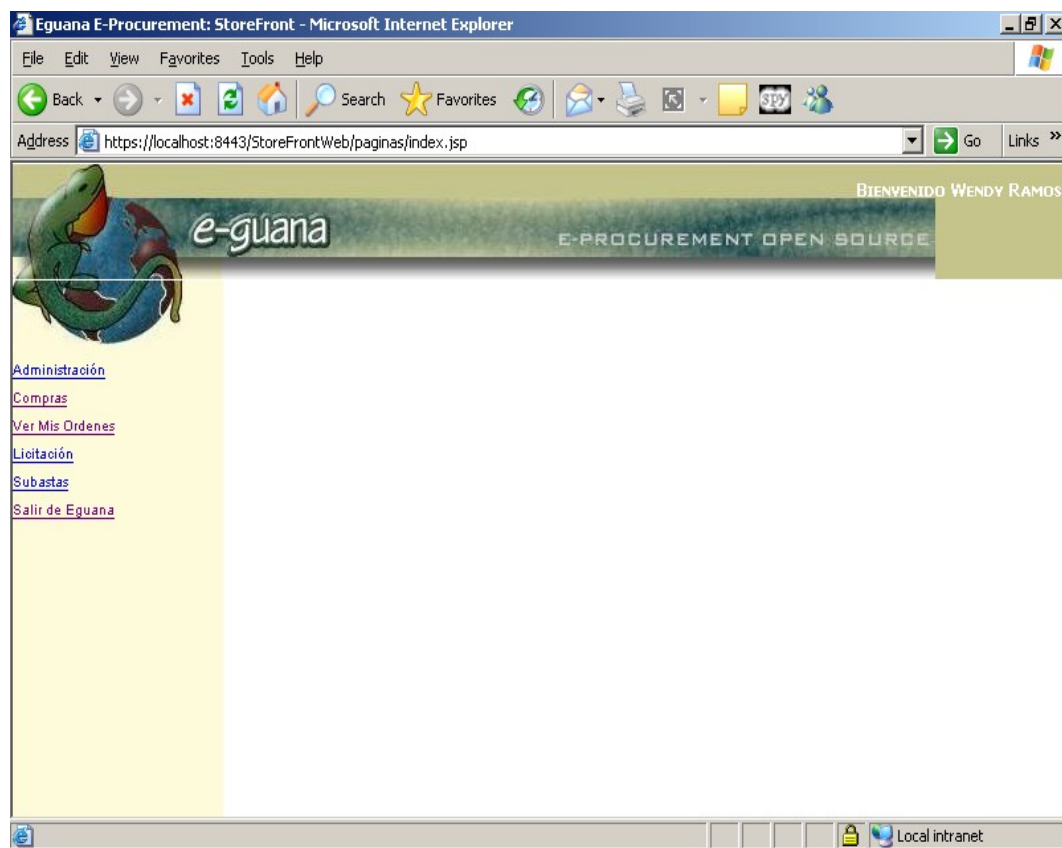
**Figura 24: Página de inicio de Eguana**

Una vez que el proceso de autenticación ha validado al usuario y tiene permiso para ingresar al sistema, se verifican qué roles tiene asignado (autorización), los cuales permitirán observar diferentes opciones del menú principal, de acuerdo a la siguiente tabla:



<b>Rol</b>	<b>Opción de Menú</b>
Comprador	Compras Ver mis Órdenes
Vendedor	Órdenes Asignadas

**Tabla 7: Roles vs. opción de menú**



**Figura 25: Página de menú de Eguana**

Una vez ingresado al sistema, sobre una conexión segura y con los permisos y roles asignados, el usuario podrá trabajar en el sistema sin inconvenientes.

***Descripción de la implementación***

Eguana, y específicamente el módulo Storefront, ha utilizado tanto la seguridad declarativa como programática para sus necesidades de seguridad. Se definirá inicialmente las configuraciones hechas a nivel del servidor y aplicación y posteriormente la personalización de las opciones del menú manejadas programáticamente.

**Dominio de seguridad**

Un dominio de seguridad especifica el nombre JNDI de la implementación de la interfase del administrador de seguridad que JBoss usa para contenedores Web y EJB, y son definidos en ciertos

descriptores. Cuando se lo define como un elemento de alto nivel dentro del descriptor de despliegue determinado, significa que afectará a todos los objetos EJBs dentro de la aplicación. Puede también definirse para un EJB específico.

Para el módulo Storefront se creó un dominio de seguridad general llamado “storefront”, el cual es almacenado en el archivo de configuración **jboss-Web.xml** que se encuentra en el directorio **WEB-INF** dentro del módulo Web de la aplicación. El contenido del archivo es el siguiente:

```
<?xml version="1.0"?>
<!DOCTYPE jboss-Web PUBLIC "-//JBoss//DTD Web Application 2.3//EN"
"http://www.jboss.org/j2ee/dtd/jboss-Web_3_0.dtd">
<jboss-Web>
  <security-domain>java:/jaas/storefront</security-domain>
  <context-root>StoreFrontWeb</context-root>
</jboss-Web>
```

Como se puede observar, el dominio de seguridad “storefront” está definido como un módulo JASS y apunta al módulo Web a través de su

nombre de contexto StoreFrontWeb, como se especifica en la línea siguiente. Como se mencionó en la sección 2.3.2.3, la implementación de seguridades en el modelo J2EE se basa en JAAS.

La implementación del administrador de seguridad de JAAS (JaasSecurityManager) en JBoss permite una personalización completa del mecanismo de autenticación por medio de la configuración del módulo de login JAAS. Definiendo la entrada de configuración del módulo de login específico, que corresponde al nombre del dominio de seguridad, se especifica el mecanismo de autenticación.

La información del dominio de seguridad se encuentra en el archivo **login-config.xml**, que está en la ruta: **%JBOSS\_HOME%\server\default\conf**. Este archivo, propio del servidor JBoss, contiene una lista de nombres de dominios de seguridad y mantiene la información del login de las diversas aplicaciones que se corren en él. Para el caso del módulo Storefront y, de acuerdo a lo

especificado en el archivo jboss-Web.xml, se ha registrado una entrada para el módulo de login de Storefront. La entrada se describe en el archivo de la siguiente manera:

```
<application-policy name="storefront">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.DatabaseServerLoginModule"
flag="required">
      <module-option name="unauthenticatedIdentity">visitante</module-option>
      <module-option name="dsJndiName">java:/MySqlDS</module-option>
      <module-option name="principalsQuery">
        SELECT clave
        FROM usuarios
        WHERE login =?
        AND estado = 'a'
      </module-option>
      <module-option name="rolesQuery">
        SELECT nombre, 'Roles'
        FROM roles
        INNER JOIN usuario_rol ON(roles.id_rol = usuario_rol.id_rol)
        INNER JOIN usuarios ON(usuarios.login = usuario_rol.login)
        WHERE usuarios.login =?
      </module-option>
    </login-module>
  </authentication>
</application-policy>
```

Como la información de usuarios y roles se encuentra almacenada en una base de datos relacional, se escogió el módulo de login del tipo **DatabaseServerLoginModule**, el cual soporta autenticación y mapeo de roles. Originalmente, este módulo trabaja con un grupo de tablas predefinidas (Principals y Roles) para obtener la información de login,

pero para el caso de Storefront se prefirió utilizar las tablas diseñadas en la base de datos de MySQL para el sistema: USUARIOS y ROLES, que mantienen la misma información: usuario, clave y roles asignados.

Las opciones configuradas son las siguientes: [22]

**unauthenticatedIdentity:**

Es el nombre que se debe asignar a peticiones que no tengan información de autenticación.

**dsJndiName:**

Es el nombre JNDI de la fuente de datos de la base de datos que contiene las tablas. La base de datos escogida para la aplicación fue MySQL.

**principalsQuery:**

Se define la sentencia SQL que consulta la información del usuario logoneado.

**rolesQuery:**

Se define la sentencia SQL<sup>1</sup> que consulta los roles asignados al usuario logoneado.

**Proceso de autenticación y autorización**

Después de la configuración de seguridades en los descriptores correspondientes, se procedió al desarrollo del proceso de autenticación basado en un formulario. Este proceso requiere los siguientes pasos:

1. Implementar la página de login.
2. Implementar la página de error si el login falla.
3. Especificar en el descriptor correspondiente el tipo de autenticación utilizado y las páginas de login y error.

---

<sup>1</sup> SQL: De sus siglas en inglés Structured Query Language.



La página de login es simple salvo los nombres de los campos del usuario, clave y nombre de la acción del formulario. Estos nombres, especificados en la definición del servlet, deben ser usados necesariamente. La tabla 3-4 presenta la lista de los nombres:

Atributo	Descripción
j_username	El nombre del campo del usuario
j_password	El nombre del campo de la clave (password)
j_security_check	El nombre de la acción del formulario de login.

**Tabla 8: Atributos del formulario de login**

El código correspondiente a la parte del login queda como se muestra a continuación:

```
<form name="fconexion" method="post" action="j_security_check" >
<table width="80%">
<tr>
<td><font face="Arial" size="1" color="black">Usuario:</font></td>
```

```
<td><input type="text" name="j_username" size="14" /></td>
</tr>
<tr>
<td><font face="Arial" size="1" color="black">Clave:</font></td>
<td><input type="password" name="j_password" size="14" /></td>
</tr>
<tr>
<td align="center">
<input type="submit" value="Conectar" name="j_security_check" />
</td>
</tr>
</table>
</form>
```

La página de error que se presentaría de existir un error durante el login únicamente contiene un enlace hacia la página de login para un nuevo ingreso.

Se finaliza la configuración de los procesos de autenticación y autorización con el registro de parámetros en el descriptor Web.xml que se encuentra en el directorio WEB-INF del módulo Web. Este archivo mantiene toda la información del módulo Web para su funcionamiento. Para el caso de los procesos mencionados, se configuró los roles de la aplicación (administrador, comprador y vendedor), el tipo de

autenticación (autenticación basado en formulario - FORM) y las páginas de login y error. Se muestra a continuación el fragmento del archivo Web.xml del módulo:

```

<security-constraint>
  <Web-resource-collection>
    <Web-resource-name>solo-para-storefront</Web-resource-name>
    <url-pattern>/paginas/*</url-pattern>
  </Web-resource-collection>
  <auth-constraint>
    <role-name>Comprador</role-name>
    <role-name>Vendedor</role-name>
  </auth-constraint>
</security-constraint>
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/paginas/login/login.jsp</form-login-page>
    <form-error-page>/paginas/errores/errorLogin.jsp</form-error-page>
  </form-login-config>
</login-config>
<security-role>
  <role-name>Comprador</role-name>
</security-role>
<security-role>
  <role-name>Vendedor</role-name>
</security-role>

```

### Implementación de página de inicio

A este nivel, la configuración declarativa de los procesos de autenticación y autorización están concluidos. Para terminar la

implementación para el ingreso de usuarios al módulo Storefront es necesario diseñar las páginas involucradas en el inicio. Como se mencionó en apartados anteriores, la página jsp que contiene el formulario de login ya ha sido diseñada pero para el ingreso al módulo se debe primero acoplar esta página a todo el conjunto del módulo Web.

El servidor JBoss comienza una aplicación Web con una página de inicio llamada index.html. En el caso del módulo Storefront, para ingresar a la aplicación el usuario primero debe iniciar la sesión con el proceso de login y éste, si es exitoso, se direccionará a la primera página del módulo. Como el proceso de login ya ha sido configurado, solo resta implementar el direccionamiento hacia la página de inicio del módulo. La página index.html se presenta de la siguiente manera:

```
<html>
<head>
  <script type="text/javascript">
    document.location = 'paginas/index.jsp';
  </script>
  <title></title>
</head>
```

```
<body></body>  
</html>
```

Como se observa, la página index.html ejecuta un direccionamiento hacia la página index.jsp, que es la página inicial del módulo Storefront, la cual una vez que el usuario inicie correctamente la sesión en el sitio, se levantará.

### **Autenticación con SSL**

Cuando se ha escogido como método de autenticación el método básico o el basado en un formulario, las claves de usuarios no son protegidas, lo cual significa que éstas son enviadas entre un cliente y un servidor en una sesión no protegida y pueden ser vistas e interceptadas por terceros. Para superar esta limitación, se pueden correr estos mecanismos sobre una sesión SSL protegida y asegurar que los datos sean protegidos para confidencialidad. Para configurar este tipo de conexión, se deben realizar dos pasos:

Especificar el requerimiento de seguridad de red en el descriptor Web.xml.

Configurar el soporte SSL en el servidor.

### **Especificación del requerimiento de red.-**

Esta especificación se realiza en el descriptor del módulo Web, Web.xml, donde se pueden escoger dos opciones: CONFIDENTIAL o INTEGRAL. La primera opción (CONFIDENTIAL) significa que cuando la aplicación requiera transmitir datos, se impida que otras entidades observen el contenido de la transmisión. La segunda opción (INTEGRAL) se define cuando la aplicación requiere que los datos sean enviados entre el cliente y el servidor de tal manera que no puedan ser cambiados en el tránsito. Aunque en principio puede existir una distinción entre estas opciones, ambas exigen el uso de SSL.

Si se escoge una de las opciones mencionadas como una limitación de seguridad, cualquiera de ellas será aplicada a todas las peticiones que

coincidan con los patrones URLs definidos como colección de recursos Web en el descriptor y no sólo con el inicio de sesión (login).

La necesidad del módulo Storefront de mantener protegida, de terceros, la información que se maneja en el sitio, escogió la opción CONFIDENTIAL, la cual exigió un cambio en el descriptor Web.xml y resultó finalmente como sigue:

```
<security-constraint>
  <Web-resource-collection>
    <Web-resource-name>solo-para-storefront</Web-resource-name>
    <url-pattern>/paginas/*</url-pattern>
  </Web-resource-collection>
  <auth-constraint>
    <role-name>Comprador</role-name>
    <role-name>Vendedor</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/paginas/login/login.jsp</form-login-page>
    <form-error-page>/paginas/errores/errorLogin.jsp</form-error-page>
  </form-login-config>
</login-config>
<security-role>
  <role-name>Comprador</role-name>
```

```
</security-role>  
<security-role>  
    <role-name>Vendedor</role-name>  
</security-role>
```

### **Configuración de soporte SSL.-**

Para instalar y configurar el soporte SSL en el servidor Web, es necesario los siguientes componentes: [3]

Certificado del servidor (keystore).

Conector HTTPS.

**Creación del certificado del servidor.-** Para usar SSL, un servidor de aplicaciones debe tener un certificado asociado por cada interfase externa o dirección IP, que acepte conexiones seguras. Un certificado de servidor es como una “licencia de manejo digital” para una dirección IP, que manifiesta la compañía con la cual el sitio está asociado y su información básica de contacto acerca de su dueño o administrador.



El certificado digital es criptográficamente firmado por su dueño y es difícil de olvidar. Para sitios involucrados en comercio electrónico u otros negocios transaccionales en los cuales la identidad es importante, un certificado puede ser adquirido de una autoridad de certificados CA<sup>1</sup> conocida.

Algunas veces la autenticación no es realmente una preocupación, por ejemplo un administrador que desea simplemente asegurarse de que el envío de datos transmitidos y recibidos por el servidor sea privado y no pueda ser “curioseado” por alguien en la conexión. En tales casos, se puede obtener un certificado firmado por si mismo (self-signed certificate). Este es el caso del módulo Storefront.

Los certificados digitales son usados con el protocolo HTTPS para autenticar clientes Web. El servicio HTTPS de la mayoría de servidores Web no correrá a menos que un certificado digital haya sido instalado.

---

<sup>1</sup> CA, Certificate Authority, tales como VeriSign o Thawte.

Una herramienta que puede ser usada para generar un certificado digital es **keytool**, una herramienta administrativa para certificados y claves que viene con J2SE SDK. Lo que hace es almacenar las claves y certificados en un archivo denominado keystore, que es un repositorio de certificados usado para identificar un cliente o un servidor. Típicamente, un archivo keystore contiene la identidad de un cliente o de un servidor, que es protegida utilizando una clave (password).

Para crear el certificado del servidor se siguen los siguientes pasos: [3]

1. Crear el archivo keystore.
2. Exportar el certificado desde el archivo keystore.
3. Firmar el certificado por una empresa CA.
4. Importar el certificado dentro de un "trust-store": un repositorio de certificados usado para verificarlos.

El archivo keystore puede ser creado en el directorio donde corre keytool, que se encuentra en la ruta %JAVA\_HOME%/bin, o en un directorio específico. La exportación del certificado se generó dentro de un archivo server.cer que es la extensión de los certificados de seguridad. La firma del certificado fue obviada porque el módulo se presentará en fase de prueba.

**Conector HTTPS.-** El conector HTTPS es configurado en el servidor JBoss, en el archivo server.xml, que está en el directorio %JBOSS-HOME%\server\default\deploy\jbossWeb-tomcat50.sar. Los parámetros especificados definen el puerto HTTPS, el certificado del servidor y atributos adicionales. Se muestra a continuación la porción del archivo correspondiente al conector configurado:

```
<Connector port="8443" address="{jboss.bind.address}" maxThreads="100"
minSpareThreads="5" maxSpareThreads="15" scheme="https" secure="true"
clientAuth="false" keystoreFile="{jboss.server.home.dir}/conf/server.keystore"
keystorePass="eguanassl" sslProtocol="TLS"/>
```

## Levantamiento del servidor

Concluida la configuración del soporte SSL, se procede a levantar el servidor JBoss, apuntando el navegador hacia la ruta `https://localhost:8443`, para comprobar el funcionamiento. Si resultó exitoso, el servidor está listo para correr la aplicación.

Una acotación importante para levantar la aplicación con el soporte SSL es que como el sitio está protegido, cuando el usuario apunta a la ruta de la aplicación con el protocolo HTTP, ésta es redireccionada al protocolo HTTPS.

### **Personalización del menú**

El usuario, si ha iniciado con éxito la sesión en el sitio, observará la pantalla de trabajo donde se muestra el menú de opciones personalizado de acuerdo a los roles que tiene asignado. Para esto se utilizó seguridad programática para manualmente implementar esta pantalla.

Se utilizaron métodos de la interfase HttpServletRequest y clases Java que obtenían la información del usuario y sus roles, lo que permitió habilitar las opciones y poder presentarlas.

### **3.1.5 Catálogo**

#### ***Antecedentes***

El catálogo de productos es la carátula de una empresa. Si decimos que el módulo StoreFront es la carátula de Eguana entonces queda en evidencia la importancia del correcto diseño e implementación del catálogo.

Por lo general el catálogo en una empresa tradicional esta compuesto de una lista de productos organizados de acuerdo a criterios como categoría o precio. El aspecto de éste varía de acuerdo a las políticas y

objetivos de la empresa, se suele a veces mostrar fotos del producto con llamativas propagandas, o simplemente una lista de precios. El catálogo electrónico tiene un potencial mucho mayor debido a que este permite el uso completo de nuevas armas como la interactividad y multimedia. Se debe tener en cuenta que el catálogo, como cualquier otra parte de un sistema e-business, debe de estar siempre alineado al propósito de una empresa.

En empresas de venta directa al consumidor los catálogos electrónicos están orientados a vender el producto, se emplean fotos, propagandas, animaciones, etc. En una empresa cuyo objetivo es facilitar la comunicación entre compradores y vendedores, el objetivo del catálogo es agilizar la búsqueda de productos para el comprador. Eguana no trata de vender el producto al comprador, por lo tanto no es necesario mostrar el producto de una manera llamativa. Por otro lado si es necesario que el usuario comprador pueda encontrar el producto de una manera rápida y tener disponible la mayor cantidad de información

sobre éste. Asumimos que el comprador entra al sistema de una manera rutinaria sabiendo de antemano que es lo que quiere comprar.

### ***Funcionalidad***

Tomando en cuenta las consideraciones definidas en los antecedentes de esta sección, se decidió incluir los siguientes aspectos en la funcionalidad:

Búsqueda de productos de acuerdo a un criterio definido por el usuario.

Búsqueda de empresas vendedoras de productos de acuerdo a un criterio definido por el usuario.

Búsqueda de categorías de productos de acuerdo a un criterio definido por el usuario.

Vista de productos por empresa.

Vista de productos por categoría.

Vista de detalles del producto.

Completa navegación dentro del catalogo entre categorías, empresas y productos.

Facilitar la comunicación con las empresas vendedoras desde el catálogo.

Fácil y rápida vista del precio del producto.

Fácil compra.

### ***Descripción de la implementación***

Para poder implementar la funcionalidad arriba descrita fue necesario tomar una decisión de diseño. Existen dos formas de implementar el catálogo en el marco de Eguana y J2EE. La primera forma es siguiendo el esquema J2EE y MVC al pie de la letra, esto es usando EJBs, servlets y JSP. El problema con este esquema es el bajo rendimiento que éste proporciona en aplicaciones como el catálogo. La naturaleza del catálogo requiere varias consultas a la base de datos en muy cortos periodos de tiempo, además el resultado de cada consulta puede traer



un gran número de registros. Los EJBs están diseñados para mantener transacciones de negocios y se desenvuelven pobremente en aplicaciones donde se deben manejar una gran cantidad de datos simultáneamente en memoria. La segunda forma de implementar el catálogo es usando directamente JDBC y clases de java simples que simulan el comportamiento de los EJBs de una manera más eficiente. Para mantenerse alineado a los objetivos de funcionalidad de Eguana, se eligió la segunda forma de implementación.

A continuación se detallan las clases que se usaron en la creación del catálogo:

<b>Nombre</b>	<b>Tipo</b>	<b>Descripción</b>
CatalogoW	Bean	Clase "wrapper" que encapsula un objeto en la tabla catálogo. Se comporta como un Bean de java.
CategoriaW	Bean	Clase "wrapper" que encapsula un objeto en la tabla categoría. Se comporta como un Bean de java.
DetalleCompraW	Bean	Clase "wrapper" que encapsula un objeto en la tabla detallecompra. Se comporta como un Bean de java.

EmpresaW	Bean	Clase “wrapper” que encapsula un objeto en la tabla empresa. Se comporta como un Bean de java.
ProductoW	Bean	Clase “wrapper” que encapsula un objeto en la tabla producto. Se comporta como un Bean de java.
EjbUtils	Clase pública	Mantiene funciones varias usadas a través de varios módulos de Eguana. Se comporta como un Bean de java.
Presentalmagen	Servlet	Obtiene la imagen de un producto desde la base de datos y la muestra directamente al navegador.

**Tabla 9: Clases del catálogo**

La mayoría de estas clases funcionan como un típico “Bean” de java, con sus respectivos “getters” y “setters” para cada campo de la tabla que representan en la base.

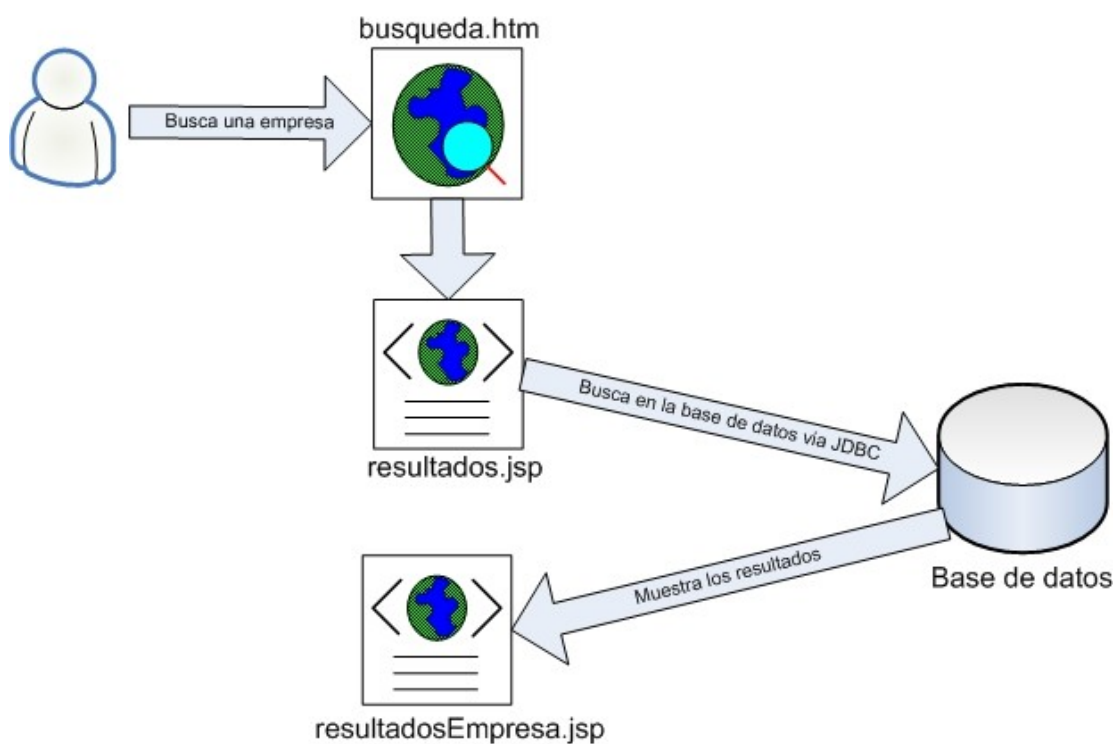
A continuación utilizamos las siguientes páginas JSP y html:

Nombre	Descripción
comprasPrincipal.htm	Pagina principal para entrar a la sección de compras.
comprasFrame.htm	Frame principal del catálogo. Contiene a

	busqueda.htm y categoriasCatalogo.jsp
busqueda.htm	Página para ingresar los criterios de búsqueda en el catálogo. Además permite ver un breve resumen de los ítems en el catálogo. Se puede buscar por descripción, categoría o empresa.
categoriasCatalogo.jsp	Muestra un resumen de los ítems en el catálogo organizados por categoría.
empresasCatalogo.jsp	Muestra un resumen de los ítems en el catálogo organizados por empresa.
catalogoEmpresa.jsp	Muestra el catálogo de una empresa.
catalogoSCategoria.jsp	Muestra el catálogo de una categoría principal.
catalogoCategoria.jsp	Muestra el catálogo de una categoría.
resultados.jsp	Procesa los requerimientos de busqueda.htm.
resultadosCategoria.jsp	Muestra los resultados de la búsqueda por categoría.
resultadosDescripcion.jsp	Muestra los resultados de la búsqueda por descripción.
resultadosEmpresa.jsp	Muestra los resultados de la búsqueda por empresa.
detallesProducto.jsp	Muestra los detalles y la imagen de un producto específico del catálogo.

**Tabla 10: Páginas del catálogo**

## Ejemplo de una búsqueda de empresas en el catálogo



**Figura 26: Funcionamiento del catálogo**

Como podemos ver en la Figura 26, la búsqueda y procesamiento de los resultados se lo hace directamente con JDBC (consultas a la base de datos) y JSP, se obvia el uso de servlets y EJBs. Esta clara violación del modelo MVC se justifica, como se menciona anteriormente, con la ganancia en rendimiento de la aplicación. En esta etapa de la interfase

con el usuario, una breve demora podría ser considerada como inaceptable. Las clases arriba mencionadas son utilizadas en las páginas JSP para procesar rápidamente las consultas retornadas por la base de datos.

### 3.1.6 **Carrito de compra**

#### ***Antecedentes***

El carrito de compras es el segundo paso en el proceso de compras luego del catálogo. El carrito debe ser lo más transparente e inocuo para el usuario, tiene que ser extremadamente fácil de usar y flexible. Su código programático debe ser lo suficientemente modular para ser reutilizado por otros módulos de Eguana.

En la mayoría de las aplicaciones e-business, el carrito de compras es una parte crucial en el diseño del proceso de compras. El carrito de compras debe siempre proveer suficiente información al usuario del estado de sus compras. Además debe estar bien claro que el carro de compras es el último paso antes de empezar la transacción y el usuario siempre debe tener la potestad de hacer y deshacer cualquier compra antes de poner su orden en el sistema.

### ***Funcionalidad***

La funcionalidad que el carrito de compra en Eguana presenta es:

Añadir productos de diferentes empresas en el catálogo a un carro de compras virtual que solo existe en la memoria del servidor de aplicaciones.

Cambiar la cantidad de ítems y borrarlos del carrito de compras.

Mostrar un total de la compra al ver el carrito de compras y durante la navegación en el catálogo.

Validar con el inventario el número de ítems de una empresa que se desea pedir.

### ***Descripción de la implementación***

Para implementar el carrito de compras usamos la metodología MVC recomendada por J2EE. El modelo es representado por la base de datos y los EJBs, más específicamente Beans de Entidad CMP. El controlador es representado por clases de java y Beans de Sesión que son usados para implementar lógica de control y de negocio. La vista es representada por páginas JSP las cuáles se encargan de presentar la información y procesan lógica de presentación.

A continuación una breve lista de los componentes del carrito de compras:

**Modelo**

Nombre	Tipo	Descripción
Catalogo	Bean de Entidad CMP	Representa una fila de la tabla catálogo.
Producto	Bean de Entidad CMP	Representa una fila de la tabla producto.
Empresa	Bean de Entidad CMP	Representa una fila de la tabla empresa.
Categoría	Bean de Entidad CMP	Representa una fila de la tabla categoría.

**Tabla 11: Clases del modelo del carro de compra****Controladores**

Nombre	Tipo	Descripción
CompraSessionBean	Stateful SessionBean	Implementa las principales acciones de control.
CarroCompra	Vector	Simulación lógica del carrito de compra en la memoria, contiene a varios objetos CompraEmpresa en su espacio de vector.
CompraEmpresa	Vector	Simulación lógica de todos los ítems de una respectiva empresa en el carrito de compras. Contiene los varios Beans de los



		productos en su espacio de vector.
EjbUtils	Clase pública	Mantiene funciones varias usadas a través de varios módulos de Eguana
CatalogoW	Bean	Clase “wrapper” que encapsula un objeto en la tabla catálogo. Se comporta como un Bean de java.
CategoriaW	Bean	Clase “wrapper” que encapsula un objeto en la tabla categoría. Se comporta como un Bean de java.
EmpresaW	Bean	Clase “wrapper” que encapsula un objeto en la tabla empresa. Se comporta como un Bean de java.
ProductoW	Bean	Clase “wrapper” que encapsula un objeto en la tabla producto. Se comporta como un Bean de java.
DetalleCompraW	Bean	Clase “wrapper” que encapsula un objeto en la tabla detallecompra. Se comporta como un Bean de java.

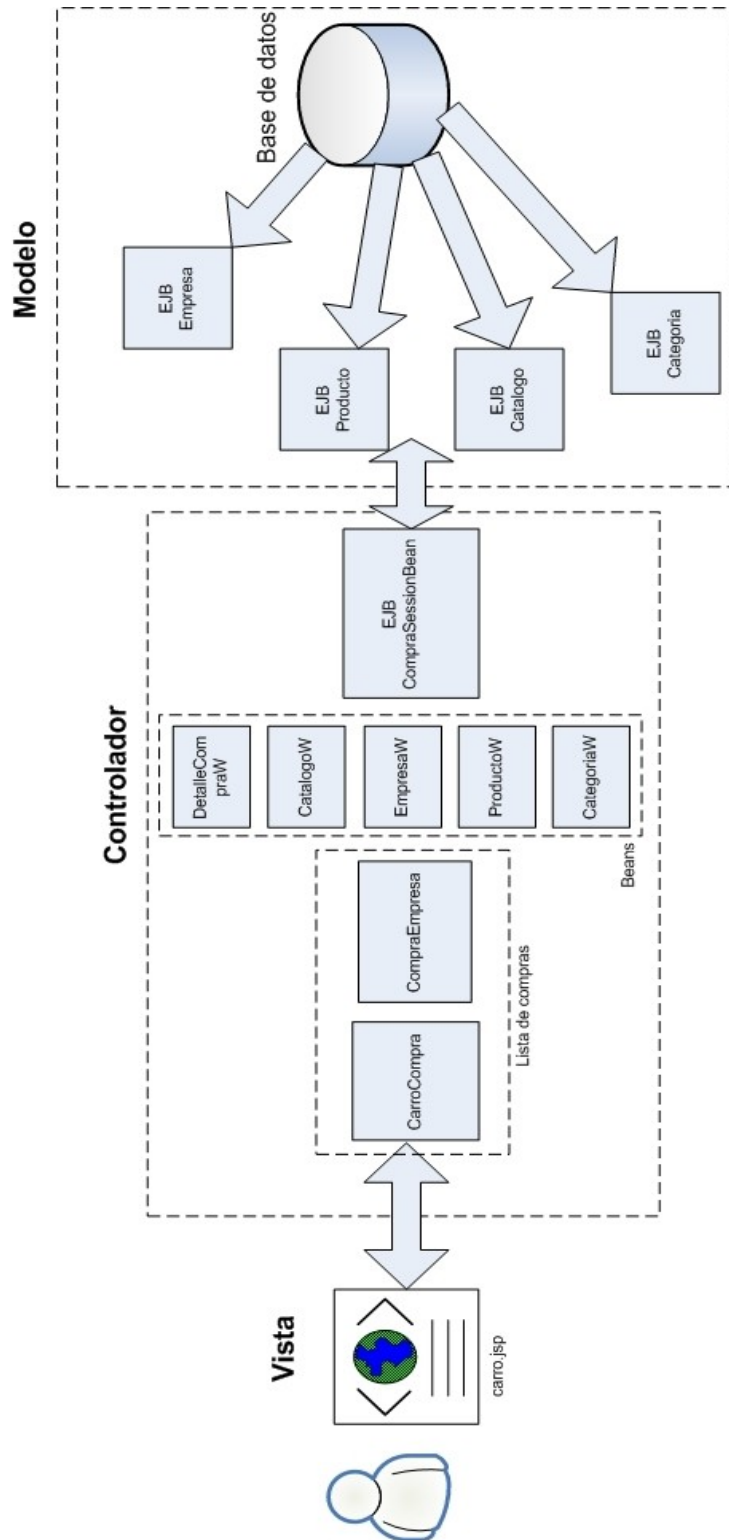
**Tabla 12: Páginas de la vista del carro de compra**

**Vista**

<b>Nombre</b>	<b>Descripción</b>
carro.jsp	Maneja la presentación del carro de compras. Valida los datos.
carroResumen.jsp	Muestra un breve resumen del total del carro de compras desde la navegación en el catálogo.
carroFinal.jsp	Muestra un breve resumen del total del carro de compras desde la navegación en el catálogo más los respectivos impuestos.

**Tabla 13: Clases controladoras del carro de compra**

## Esquema de funcionamiento del carrito de compras



**Figura 27: Funcionamiento del carrito de compras**

Revisando la Figura 27 podemos apreciar que en el diseño del carro de compras se encuentran claramente divididas las secciones del modelo, controlador y vista. El diseño tradicional implica el uso de servlets como controlador. En este diseño se ha ido más allá y se ha usado a un Bean de Sesión como controlador principal debido a que este provee una mayor flexibilidad y es una tecnología superior a servlets. Cabe recalcar que en el gráfico se han obviado algunos detalles para la mejor comprensión de éste, por ejemplo la página carro.jsp además de hacer uso de la clase vector CarroCompra también hace un uso extensivo de los Beans, obtenidos a través de CarroCompra y CompraEmpresa.

### 3.1.7 **Proceso de compra**

#### ***Antecedentes***

Los procesos de compras automatizados es lo que se denomina los sistemas e-procurement, por lo cual proveen un interfaz Web donde se visualiza el catálogo de los productos que permite a los compradores seleccionar o configurar sus requisiciones, las cuales son almacenadas en un carro de compras, la etapa final en la gestión del proceso de compra de uno o varios productos es la generación de la orden, cuando el comprador decide aceptar el carro de compra genera la orden que es registrada en el sistema.

Generalmente las aplicaciones e-procurement integran con el sistema de gestión de la empresa el proceso de control y gestión de pagos. Los controles de pagos varían de acuerdo a las políticas internas definidas en la empresa. Por esta razón Eguana consideró sumamente necesaria la implementación de la simulación del proceso de gestión de pagos lo que se define como el módulo bancario, el cual es una simulación de una aplicación bancaria real.

**Funcionalidad**

El usuario interesado en realizar el proceso de compra en el sistema EGUANA debe estar previamente registrado en el mismo, lo cual lo puede realizar desde el módulo de ADMINISTRACIÓN, posteriormente a esto debe ingresar al módulo STOREFRONT el cual le permitirá realizar el proceso de compra deseado.

El proceso de Compras en el sistema considera los siguientes puntos en su funcionalidad:

Ingresar en el módulo StoreFront.

Escoger la opción compras, la cual le presentará un catálogo de productos registrados en el sistema.

Escoger del catálogo de productos los que desea adquirir, los cuales se encuentran categorizados por descripción, categoría y empresa.

Añadir al carro de compras los productos que desee adquirir.

Confirmar los productos seleccionados y que se encuentran en el carro de compras del sistema.

Realizar la orden de compra de los productos que han sido seleccionados por el usuario que ha realizado la compra en el sistema EGUANA.

Crear una orden de compra por cada una de las empresas en las que se estén seleccionando los productos.

Enviar un mensaje al Banco para realizar los respectivos débitos correspondientes a los valores de las órdenes realizadas.

Asignar un vendedor a cada orden de compra.

Emitir notificación de e-mail a cada usuario vendedor.

### ***Descripción de la implementación***

El proceso de compras involucra transacciones con dos tablas importantes de la base de datos de StoreFront: la tabla compra y la tabla detalle\_compra.

Para acceder a la información de la base de datos y para hacer operaciones de negocio se implementaron varios componentes J2EE incluyendo Beans de sesión, de entidad y dirigidos a mensajes además de clientes Web que utilizan páginas JSP.

Se listan los objetos implementados en el proceso de generación de la orden de compra.

<b>Objeto</b>	<b>Nombre</b>
Bean de sesión sin estado	OrdenCompraSessionBean
Bean de entidad	CompraBean (CMP)
Bean de entidad	DetalleCompraBean (CMP)

**Tabla 14: Objetos del proceso de compra**



El Bean de sesión `OrdenCompraSessionBean` tipo `Stateless` tiene implementada la lógica que permite recuperar los usuarios que poseen el rol `vendedor` asociados a la empresa a la cual se le está realizando la compra (`getUsuarioVendedor`), esto se realiza por medio de consultas `jdbc` a la base de datos.

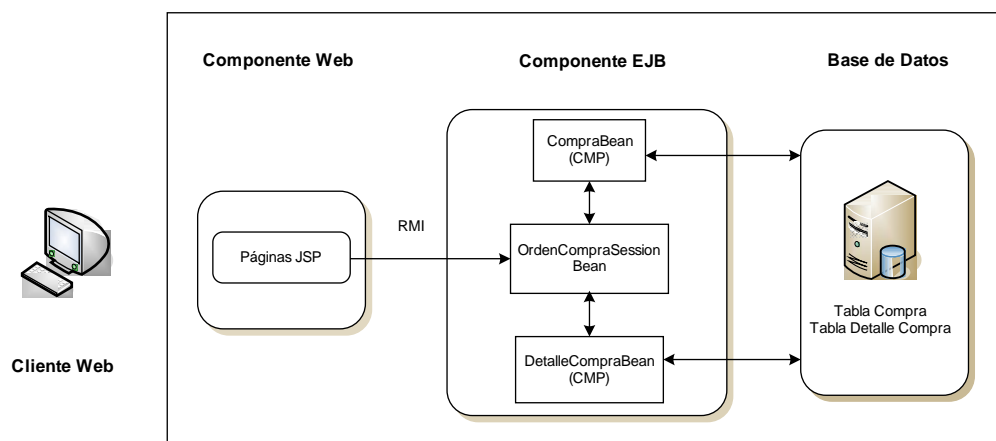
Adicionalmente permite recuperar los datos requeridos para generar la orden de compra y su detalle (Información que se obtiene desde el carro de compras), almacenándolos en las respectivas tablas:

Tabla `compra`, registra información de la compra por empresa así como también su respectivo estado (`E` que indica que la orden se encuentra en `Proceso`).

Tabla `detalle_compra`, almacena la información de los productos que se han comprado.

Para registrar la información en las tablas se utiliza los Beans de entidad tipo `CMP CompraBean` y `DetalleCompraBean`. Cada una de

estas clases funciona como un típico Bean, con sus respectivos “getters” y “setters” para cada campo de la tabla que representan en la base.



**Figura 28: Diagrama del proceso de compra**

Como podemos visualizar en la Figura 28, el cliente Web, envía una petición por medio de la página ordenes.jsp a los Beans utilizando RMI (Remote Method Invocation).

La práctica usual para acceder a los métodos de negocio de Beans CMP, es utilizar un Bean de sesión, que encapsule la lógica de negocio y actúe como interface para otros componentes EJB. En este caso se puede acceder a CompraBean y DetalleCompraBean mediante OrdenCompraSessionBean.

Los Beans de entidad implementados son tipo CMP, es decir el contenedor maneja la comunicación entre los Beans y la base de datos, lo que es una característica muy poderosa.

### **Comunicación en el proceso de compras**

Todo proceso de compra en un sistema e-procurement, como se ha podido describir en capítulos anteriores, se fundamenta en determinadas etapas que brindan rapidez y eficiencia al proceso.

Estas etapas son el pago de las órdenes de compra generadas, ya descrito, y el seguimiento que se da a las mismas hasta llegar a su término exitoso, que finaliza con la etapa de despacho. Durante estas etapas, cabe recalcar la importancia que tiene la adecuada comunicación entre los usuarios y el sistema, por ejemplo, en las siguientes tareas:

Mantener informado a los usuarios de las empresas compradoras del estado de sus compras realizadas.

Mantener informado a los usuarios de las empresas vendedoras del estado de las compras que les fueron asignadas.

Realizar las operaciones de débito/crédito en la aplicación bancaria dependiendo del saldo de la cuenta de la empresa compradora.

Para que estas etapas puedan efectuarse con la velocidad deseada y, sobre todo, que brinden el tipo de comunicación que se requiere, se vio la necesidad de utilizar tecnologías de mensajería y de envío de correo electrónico que permitan esta comunicación dentro del proceso de compras.

En el módulo Storefront se utiliza la tecnología de mensajería para la comunicación con la aplicación bancaria, en la cual Storefront envía y recibe mensajes de esta aplicación para comprobar si es posible realizar la transacción dependiendo del saldo de la cuenta de la empresa compradora y también para evaluar si es posible anular la compra por decisión del usuario comprador. La mensajería se ajusta

para esta funcionalidad eficazmente y permite una comunicación entre los módulos de manera asincrónica, que es lo que se desea alcanzar.

Para que esta funcionalidad se adapte a la plataforma J2EE, se utilizó el API de Mensajería de Java, JMS.

### **Implementación del API de mensajería JMS en el proceso de compras.-**

Debido a la funcionalidad requerida para el proceso de comunicación con la aplicación bancaria (un solo consumidor y sin dependencia de tiempo), se escogió el dominio punto a punto para la forma de comunicación y un tipo de consumo asincrónico, para lo cual se utilizó como oyentes de los mensajes a Beans manejados por mensajes (Message-Driven Bean).

Las clases creadas se describen a continuación:

### 3-11 C

Clase	Tipo	Descripción
DatosMensaje	Clase Java Serializable	Clase que mantiene los datos para la transacción bancaria.
EnviaMensaje	Clase Java Serializable	Clase que envía el mensaje a la cola específica.
MensajeBean	Bean manejado por mensajes	Actúa como oyente de los mensajes enviados por el modulo bancario y que, de acuerdo al resultado emitido, ejecuta el cambio de estado de la orden de compra.

**Tabla 15: Clases para mensajería en Storefront**

Estas clases trabajan en conjunto con las clases de la aplicación bancaria para establecer la comunicación y ejecutar las operaciones requeridas. En ellas se definió código XDoclet que realiza la generación de los datos de configuración para mensajería en los descriptores del servidor JBoss, como por ejemplo el repositorio de mensajes de la aplicación bancaria, el nombre del receptor de los mensajes, etc. Entre los parámetros configurados constan los siguientes:

Parámetro	Descripción
Tipo de Destino	javax.jms.Queue

Modo de Reconocimiento	Auto-acknowledge
Tipo de Transacción	Container
Destino JNDI Storefront	queue/RespuestaBanco
Destino JNDI Aplic. Bancaria	queue/Banco

**Tabla 16: Parámetros en descriptores XML para mensajería**

De acuerdo a los parámetros declarados, se generó entradas en los siguientes archivos descriptores:

### **jboss.xml**

```

<message-driven>
  <ejb-name>MensajesABanco</ejb-name>
  <destination-jndi-name>queue/Banco</destination-jndi-name>
</message-driven>
<message-driven>
  <ejb-name>Mensaje</ejb-name>
  <destination-jndi-name>queue/RespuestaBanco</destination-jndi-name>
</message-driven>

```

### **ejb-jar.xml**

```

<message-driven >
  <description><![CDATA[Receptor de mensajes para el
banco]]></description>
  <display-name>Receptor de mensajes para el banco</display-name>
  <ejb-name>MensajesABanco</ejb-name>
  <ejb-class>org.eguana.banco.MensajesABancoBean</ejb-class>
  <transaction-type>Container</transaction-type>
  <acknowledge-mode>Auto-acknowledge</acknowledge-mode>

```



```

<message-driven-destination>
  <destination-type>javax.jms.Queue</destination-type>
  <subscription-durability>NonDurable</subscription-durability>
</message-driven-destination>
</message-driven>

<message-driven >
  <description><![CDATA[]]></description>
  <ejb-name>Mensaje</ejb-name>
  <ejb-class>org.eguana.storefront.mdb.ejb.MensajeBean</ejb-class>
  <transaction-type>Container</transaction-type>
  <acknowledge-mode>Auto-acknowledge</acknowledge-mode>
  <message-driven-destination>
    <destination-type>javax.jms.Queue</destination-type>
  </message-driven-destination>
</message-driven>

```

Como se observa en los descriptores, cada módulo mantiene una cola que es la que contiene los mensajes generados y la que recibe los emitidos por el otro módulo. Estas son asociadas a su respectivo Bean de mensajes con el cual trabajan.

### **Uso de correo electrónico en el proceso de compras**

Después de la generación de las órdenes de compra, el sistema mantiene informado al comprador y vendedor sobre los estados por los

que pasa la orden de compra, a través de una aplicación de correo electrónico implementado en el módulo. Para desarrollarlo se utilizó el API de Java para Correo Electrónico: JavaMail.

### **Implementación del API Java Mail en el proceso de compras.-**

Como la forma de comunicación más común de los portales Web con sus usuarios es a través del correo electrónico, se utilizó el API de JavaMail que nos brinda toda la especificación para implementar un sistema de mail.

La implementación de un sistema de mail con el API JavaMail es sencilla. Para el efecto se creó únicamente una clase, `EnviaMail.java`, que contiene el método que envía el correo electrónico y se creó un descriptor xml en el servidor JBoss que hace referencia a la configuración del sistema.

Este archivo debe estar localizado en la ruta %JBOSS\_HOME%\server\default\deploy y, para el caso del proyecto, recibió el nombre de eguana-mail-service.xml. Este archivo reemplazó el descriptor xml existente para configuración de correo: mail-service.xml. Los parámetros configurados en este archivo se muestran a continuación:

```
<mbean code="org.jboss.mail.MailService" name="jboss:service=Mail">
  <attribute name="JNDIName">java:/EguanaMail</attribute>
  <attribute name="Configuration">
    <!-- Test -->
    <configuration>
      <!-- Change to your mail server protocol -->
      <property name="mail.transport.protocol" value="smtp"/>
      <!-- Change to the SMTP gateway server -->
      <property name="mail.smtp.host" value="mercurio.cedia.ec"/>
      <!-- Change to the address mail will be from -->
      <property name="mail.from" value="admin@empresa.com"/>
      <!-- Enable debugging output from the javamail classes -->
      <property name="mail.debug" value="true"/>
    </configuration>
  </attribute>
</mbean>
```

Como se observa, los parámetros son fácilmente configurables, así por ejemplo se muestra el nombre JNDI con el cual será reconocido el servicio de mail: java:/EguanaMail, el nombre del protocolo a usar: SMTP<sup>1</sup>, el nombre del servidor SMTP, la dirección de correo electrónico remitente, etc.

### 3.1.8 Proceso de despacho

#### ***Antecedentes***

Las aplicaciones E-procurement no solo abarcan la gestión del proceso de compras y los controles de pago, sino que implementan también una funcionalidad muy importante y vital en el momento del diseño de aplicaciones que gestionan compras en internet como es el proceso de despacho.

---

<sup>1</sup> SMTP: Simple Mail Transport Protocol. Usado para enviar e-mail sobre Internet.

Este proceso varía dependiendo de las políticas internas implementadas en las empresas. Dichas políticas son las que definen los controles que se deben realizar en la aplicación.

### ***Funcionalidad***

Al gestionar el usuario del sistema una orden de compra, la misma es asignada a un vendedor de la empresa a la cual se adquirió el o los productos, generando una orden en el sistema en estado en proceso (E), la cual está siendo procesada por el banco para realizar los respectivos débitos de las cuentas que pertenecen a los usuarios que realizan la compra.

Al terminar de ser procesada la orden por el banco ésta puede tener dos posibles estados:

Estado	Descripción
N	Orden no aprobada
P	Orden Pagada

**Tabla 17: Estados de las órdenes de compra**

Dependiendo del estado en el que se encuentre la orden el usuario vendedor asignado podrá ejecutar o no el proceso de despacho.

El proceso de despacho de órdenes de compra en el sistema considera los siguientes puntos en su funcionalidad:

Ingresar en el módulo StoreFront.

Escoger la opción Ver órdenes asignadas.

Seleccionar el estado de las órdenes de compra que desea visualizar, el cual debe ser pagada (P) para el proceso de despacho.

Escoger las órdenes de compra que desea despachar.

Emitir un email automáticamente a los usuarios compradores de las ordenes de compras despachas, indicándoles que sus respectivas ordenes han sido procesadas.

### ***Descripción de la implementación***

El proceso de despacho involucra transacciones con la tabla compra del modelo de la base de datos StoreFront.

El proceso de despacho es ejecutado por los usuarios vendedores asignados a cada orden. Para realizar este proceso el usuario correspondiente deberá ingresar al sistema y podrá visualizar un link que le permitirá consultar las órdenes que le han sido asignadas con sus respectivos estados. Este proceso sólo se podrá ejecutar en aquellas órdenes que se encuentren en estado pagada (P).

Para implementar este proceso se utilizó el framework Sofia que es una herramienta basada en el principio modelo, vista y controlador (MVC).

Se listan los objetos implementados en los procesos de despacho y cancelación de las órdenes de compra.

<b>Objeto</b>	<b>Nombre</b>
Modelo	ConsultaOrdenes.java
Vista	consultaDetalle.jsp
Controlador	Control.java

**Tabla 18: Objetos del proceso de despacho**

En un framework como Sofia, se necesita generalmente estos tres componentes para constituir una página Web compleja.

La clase modelo le especifica a la página una serie de etiquetas para conseguir los datos que se deben presentar en la consulta solicitada



dependiendo si la consulta es realizada por el usuario comprador o vendedor y el estado de las órdenes que desee visualizar. Existe un atributo en la página que indica que los datos vendrán de una consulta SQL y que deben recuperarse automáticamente de la base de datos cada vez que se consulte la página. Es importante resaltar que este framework Sofia utiliza JDBC para realizar las consultas y obtener los datos de la base.

La página JSP posee la lógica de la presentación de la consulta. Es la encargada de invocar el datasource<sup>1</sup>, en este atributo se realiza la llamada a la clase modelo que recupera la información a visualizar de la base de datos.

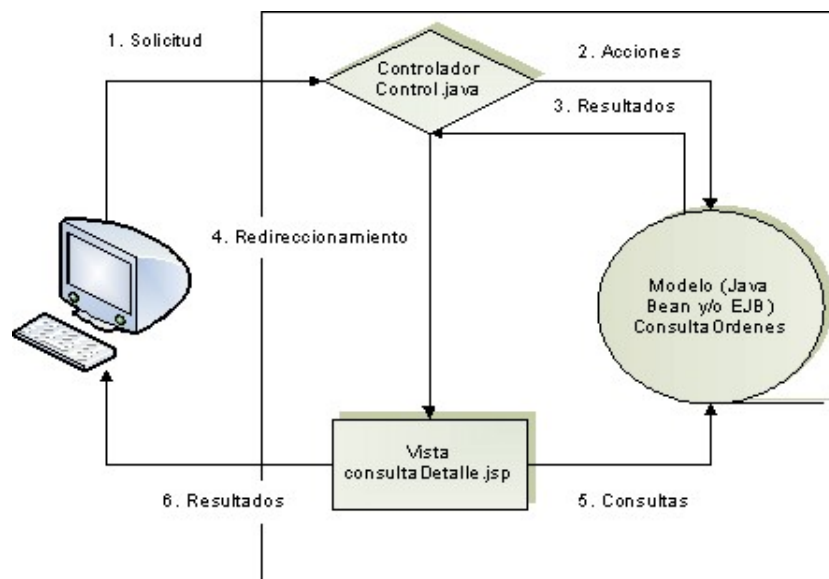
La clase controlador es la que responderá al proceso que tiene implementado el botón Grabar, posee la lógica de control para determinar que acción debe ejecutarse dependiendo del requerimiento que se solicitó desde la página JSP. El controlador posee las

---

<sup>1</sup> Datasource: se refiere al origen de los datos.

condiciones que se deben filtrar para que se ejecuten las sentencias

SQL correspondientes para el proceso de despacho.



**Figura 29: Estructura del patrón MVC**

El proceso de despacho debe enviar automáticamente un email al comprador, para lo cual se implementaron clases que utilizan los APIs de java de mensajería y correo electrónico como se explicó en la sección del proceso de compra.

### 3.1.9 **Proceso de cancelación**

#### ***Antecedentes***

En la mayoría de las aplicaciones e-business el proceso de cancelación es una de las flexibilidades que se proporcionan al usuario comprador, dándole al mismo la oportunidad de reversar o cancelar su compra.

Al igual que el proceso de despacho, este proceso también varía dependiendo de las políticas internas implementadas en las empresas, las cuales definen los controles que se deben realizar en la aplicación.

### ***Funcionalidad***

Al igual que el proceso de despacho, el proceso de cancelación debe considerar el estado en el que se encuentra la orden de compra y dependiendo de este el usuario comprador o vendedor podrá ejecutar o no el proceso de cancelación.

El vendedor podrá cancelar la orden en el caso de no existir en stock los productos que conforman la orden.

El proceso de cancelación de órdenes de compra en el sistema considera los siguientes pasos en su funcionalidad:

Ingresar en el módulo StoreFront.

Escoger la opción ver ordenes de compra.

Seleccionar el estado de las órdenes de compra que desea visualizar, para el proceso de cancelación los estados de las ordenes pueden ser: en proceso (E) o pagada (P). Si la orden que se desea anular no es visualizada en el listado que se presenta en el estado e o p, significa que la misma ya ha sido despacha por el usuario vendedor y por ende no se podrá anular.

Escoger las órdenes de compra que desea anular.

Emitir mensajes al banco para notificar que se desea anular la orden de compra por lo cual no se deberá ejecutar el débito en la cuenta del usuario comprador.

Enviar email automáticamente de confirmación a vendedor y comprador informándoles que la orden de compra ha sido cancelada.

### ***Descripción de la implementación***

El proceso de cancelación de una orden, al igual que el proceso de despacho, realiza actualizaciones en la información de la tabla compra del modelo de la base de datos StoreFront.

Este proceso es ejecutado por los usuarios compradores o vendedores. Para ejecutar este proceso el usuario debe ingresar al sistema y podrá visualizar un link que le permitirá consultar las órdenes de las compras que ha realizado o le han sido asignadas respectivamente, seleccionando el estado de las órdenes que desea visualizar.

Como se indicó en el proceso de despacho para la implementación de este proceso se ha utilizado el framework Sofia.

Los objetos implementados son los mismos que se han especificado en el proceso de despacho, lo que si es imprescindible recalcar, es que dependiendo del proceso que se ejecuta se realizan las validaciones y controles requeridos.

El proceso de cancelación como se indicó anteriormente se puede ejecutar sólo en ordenes con estado en proceso (E) o en estado pagada (P), dependiendo de esto el proceso interactúa con la simulación del módulo bancario que se encarga de enviar los mensajes correspondientes.

Si el estado de la orden es en proceso, se envía el mensaje al banco de que no realice el débito, puesto que se cancelará la orden, caso contrario si el estado es pagada, se envía el mensaje al banco para que reverse el pago que se realizó, ya que la orden será cancelada.

Además este proceso debe enviar automáticamente un email para comunicar a los usuarios compradores y vendedores que se ha cancelado la orden, para lo cual se implementaron clases que utilizan los APIs de java de mensajería y correo como se explicó en la sección del proceso de compra.

## **3.2 Diseño**

El diseño implementado en la aplicación es basado en una solución J2EE donde convergen una serie de elementos fundamentales: metodología de desarrollo, sesiones de análisis y diseño orientado a objetos, y sobre todo el uso del software libre para la codificación.

La conjunción de todos estos elementos han hecho económicamente accesible la implantación de la plataforma J2EE y viable el desarrollo de la misma. Es importante recalcar que resulta una solución económica y completamente segura.

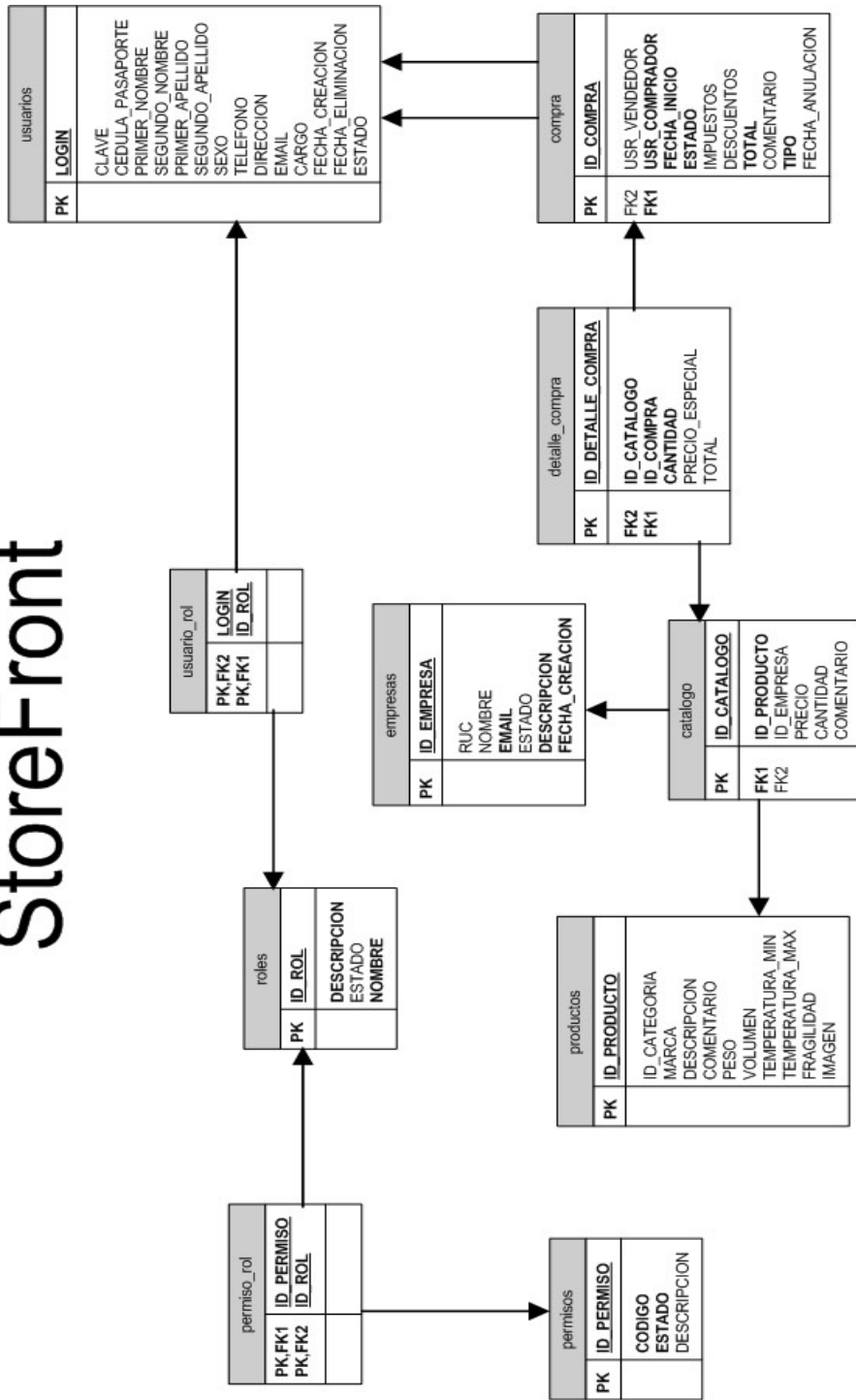
### **3.2.1 Base de datos**

#### **3.2.1.1 Diagrama entidad relación**



Abajo tenemos un gráfico de la base de datos del módulo StoreFront:

# StoreFront



### **Figura 30: Diagrama entidad relación**

#### **3.2.1.2 Descripción de la base de datos**

El esquema de base de datos del módulo StoreFront está compuesto por diez tablas:

La tabla usuarios almacena información de los usuarios que se encuentran registrados en el sistema Eguana, los cuales pueden ingresar al módulo StoreFront y realizar las transacciones que este módulo ofrece.

La tabla usuario\_rol almacena información de los roles que están asociados a los usuarios registrados en el sistema.

La tabla permisos registra información de los permisos que existen configurados para la aplicación Eguana (Por ejemplo: comprar, vender, etc.).

La tabla permiso\_rol registra información de las acciones configuradas en el sistema y que pueden ser ejecutadas por los diferentes roles que existen.

La tabla roles posee información sobre los roles que existen configurados en el sistema.

La tabla empresas posee información de las empresas que se encuentran registradas en el sistema.

La tabla productos mantiene un inventario de los productos disponibles en Eguana.

La tabla catalogo posee información de los productos asociados a las respectivas empresas registradas en el sistema.

La tabla compra almacena el registro de las compras que han sido realizadas en el sistema por los diferentes compradores.

La tabla detalle\_compra almacena el detalle de las compras realizadas en el sistema.

### 3.2.2 Pantallas del sistema



Figura 31: Pantalla de inicio de sesión

Eguana E-Procurement: StoreFront - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

https://localhost:8443/StoreFrontWeb/paginas/index.jsp

**e-guana** E - P R O C U R E M E N T O P E N S O U R C E

Bienvenido Wendy Ramos Lunes, 21 de Noviembre de 2005 9:17:48 PM

**Compras | e-guana**

**Búsqueda de Productos**

Descripción   
 Empresa   
 Categoría

**ELECTRODOMESTICOS**  
Televisores, Microondas, ...

**LIBROS**  
Ciencia Ficción, ...

**COMPUTADORAS**  
Tarjetas de red, Comp Compaq, ...

**TELECOMUNICACIONES**  
Radios Microondas, ...

**VÍVERES**  
Lácteos, Frutas, ...

**MÚSICA**  
CDs, ...

**PELÍCULAS**  
DVDs, ...

**SOFTWARE**  
Video juegos, Software Oficina, Software Enterprise, ...

**Nuestras Empresas**

Existen diversas empresas que han se han decidido por Eguana a la hora de establecer sus relaciones comerciales con otras empresas, entre las que figuran empresas de:

- Electrodomésticos
- Papelería y Oficina
- Supermercados
- Librerías
- Música
- Hardware y software
- Telecomunicaciones
- Construcción, etc.

Entre las empresas que nos acompañan están:

- Comandato
- Alm. Japón
- Supermaxi

**Inicio**

**Compras**

**Compras Realizadas**

**Licitación**

**Subastas**

**Salir de Eguana**

**Promociones**

**supermaxi**  
Ofertas en todos sus productos por aniversario.

**Juan Marcet**  
Promoción de lámparas

Intranet local

Figura 32: Sección de compras con el catálogo

The screenshot shows the 'Compras' section of the e-guana system. The browser window is titled 'Eguana E-Procurement: StoreFront - Microsoft Internet Explorer'. The page header includes the e-guana logo and the text 'E-PROCUREMENT OPEN SOURCE'. A navigation menu on the left lists 'Inicio', 'Compras', 'Compras Realizadas', 'Licitación', 'Subastas', and 'Salir de Eguana'. A 'Promociones' box highlights a promotion for 'Juan Marcet' regarding desk lamps. The main content area features a search bar with the text 'Búsqueda de Productos' and a 'Buscar' button. Below the search bar, there are radio buttons for 'Descripción', 'Empresa', and 'Categoría'. The product catalog is titled 'Electrodomesticos > Televisores' and is organized into three sections: 'Comandato', 'Almacenes Japon', and 'Supermaxi'. Each section contains a table with columns for 'Descripción', 'Precio', and 'Comprar'.

**Electrodomesticos > Televisores**

Descripción	Precio	Comprar
Televisor Samsung 21" Modelo 34G	100.00	[Comprar]
Televisor LG 21" Modelo TriMaster200	112.00	[Comprar]
Televisor Sony Flatron 25	115.00	[Comprar]
Televisor Panasonic ViewMe Encore	123.00	[Comprar]
Televisor Emerson 1500	233.00	[Comprar]

**Almacenes Japon**

Descripción	Precio	Comprar
Televisor LG 21" Modelo TriMaster200	125.00	[Comprar]
Televisor Sony Flatron 25	68.00	[Comprar]

**Supermaxi**

Descripción	Precio	Comprar
Televisor Samsung 21" Modelo 34G	101.00	[Comprar]
Televisor LG 21" Modelo TriMaster200	113.00	[Comprar]
Televisor Sony Flatron 25	111.00	[Comprar]
Televisor Panasonic ViewMe Encore	124.00	[Comprar]
Televisor Emerson 1500	250.00	[Comprar]

**Nuestras Empresas**

Existen diversas empresas que han se han decidido por Eguana a la hora de establecer sus relaciones comerciales con otras empresas, entre las que figuran empresas de:

- Electrodomesticos
- Papelaria y Oficina
- Supermercados
- Librerias
- Musica
- Hardware y software
- Telecomunicaciones
- Construcción, etc.

Entre las empresas que nos acompañan están:

- Comandato
- Alm. Japon
- Supermaxi
- Juan Marcet
- Libreria Cientifica
- y muchas más.

Figura 33: Catálogo de productos





Eguana E-Procurement: Storefront - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

https://localhost:8443/StoreFrontWeb/paginas/index.jsp

Bienvenido Wendy Ramos Lunes, 21 de Noviembre de 2005 9:19:50 PM

**Compras | e-guana**

Búsqueda de Productos

Descripción Empresa Categoría

**Carrito de Compras**

**Empresa: Comandato**

Descripción	Cantidad	Precio	Total
Televisor Samsung 21" Modelo 34G	1	100.00	100.00
Total Empresa: 100.00			

**Empresa: Libreria Cientifica**

Descripción	Cantidad	Precio	Total
Robert A. Henlein - Stranger in a Strange Land	12	11.00	132.00
Total Empresa: 132.00			

**Empresa: GenSystems**

Descripción	Cantidad	Precio	Total
Palm Tungsten T5	1	350.00	350.00
Total Empresa: 350.00			

**Total:** 582.00

CONTINUAR ORDENAR

Total Carrito \$ 582.00

Ver Carrito

Promociones

Televisores de pantalla plana LG, Sony a precios rebaja.

Almacenes Japón con garantía de exportación.

Listo Intranet local

**Figura 34: Carrito de compras**

Eguana E-Procurement Storefront - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

Dirección https://localhost:8443/StoreFrontWeb/paginas/index.jsp

**e-guana** E - PROCUREMENT OPEN SOURCE

Bienvenido Wendy Ramos Lunes, 21 de Noviembre de 2005 9:20:15 PM

**Compras | e-guana**

**Búsqueda de Productos**

Descripción Empresa Categoría

**¿ Desea ordenar el carrito de compras ?**

Descripción	Cantidad	Precio	Total
Empresa: Comandato			
Televisor Samsung 21" Modelo 34G	1	100.00	100.00
Total Empresa: 100.00			
Empresa: Librería Científica			
Robert A. Henlein - Stranger in a Strange Land	12	11.00	132.00
Total Empresa: 132.00			
Empresa: GenSystems			
Palm Tungsten T5	1	350.00	350.00
Total Empresa: 350.00			

<b>Sub Total</b>	582.00
<b>IVA:</b>	69.84
<b>Total:</b>	651.84

SI NO

**Total Carrito**  
\$ 582.00

Ver Carrito

**Nuestras Empresas**

Existen diversas empresas que han se han decidido por Eguana a la hora de establecer sus relaciones comerciales con otras empresas, entre las que figuran empresas de:

- Electrodomésticos
- Papelaría y Oficina
- Supermercados
- Librerías
- Música
- Hardware y software
- Telecomunicaciones
- Construcción, etc.

Entre las empresas que nos acompañan están:

- Comandato
- Alm. Japón
- Supermaxi
- Juan Marcat
- Librería Científica
- y muchas más.

Inicio  
Compras  
Compras Realizadas  
Licitación  
Subastas  
Salir de Eguana

Promociones

**Almacenes Japón**  
Ollas con garantía, de exportación.

**Supermaxi**  
Ofertas en todos sus departamentos.

Listo Intranet local

Figura 35: Confirmación del carrito de compras

Eguana E-Procurement: StoreFront - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

Atrás Búsqueda Favoritos

Dirección https://localhost:8443/StoreFrontWeb/paginas/index.jsp

**e-guana** E-PROCUREMENT OPEN SOURCE

Bienvenido Wendy Ramos Lunes, 21 de Noviembre de 2005 9:21:18 PM

**Compras Realizadas** | e-guana

Estado:

Compra	Fecha Creación	Estado	Categoría	Producto	Total	ANULAR ORDEN
922 - GenSystems	21 de noviembre 2005	EN PROCESO			\$392.0	<input type="checkbox"/>
			Palm	1 Palm Tungsten T5	\$350.0	<input type="checkbox"/>
921 - Librería Científica	21 de noviembre 2005	EN PROCESO			\$147.84	<input type="checkbox"/>
			Ciencia Ficción	12 Robert A. Heinlein - Stranger in a Strange Land	\$132.0	<input type="checkbox"/>
920 - Comandato	21 de noviembre 2005	EN PROCESO			\$112.0	<input type="checkbox"/>
			Televisores	1 Televisor Samsung 21" Modelo 34G	\$100.0	<input type="checkbox"/>

**Inicio**  
Compras  
Compras Realizadas  
Licitación  
Subastas  
Salir de Eguana

**Promociones**

Ollas con garantía, de exportación.

**Supermaxi**  
Ofertas en todos sus productos por aniversario.

**Nuestras Empresas**

Existen diversas empresas que han se han decidido por Eguana a la hora de establecer sus relaciones comerciales con otras empresas, entre las que figuran empresas de:

- Electrodomésticos
- Papelería y Oficina
- Supermercados
- Librerías
- Música
- Hardware y software
- Telecomunicaciones
- Construcción, etc.

Entre las empresas que nos acompañan están:

- Comandato
- Alm. Japón
- Supermaxi
- Juan Marcat
- Librería Científica
- y muchas más.

Listo Intranet local

**Figura 36: Consulta de órdenes**

Eguana E-Procurement: StoreFront - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

Atrás Búsqueda Favoritos Ir Vínculos

Dirección https://localhost:8443/StoreFrontWeb/paginas/index.jsp

**e-guana** E - P R O C U R E M E N T O P E N S O U R C E

Bienvenido Roberto Guerrero Lunes, 21 de Noviembre de 2005 9:23:10 PM

**Anuncio de Eguana:**  
Le ha sido asignada una orden nueva en las últimas 24 horas!

Eguana E-Procurement es un portal que integra en un solo lugar a empresas de cualquier índole, que deseen mantener y mejorar sus relaciones comerciales, con sólo ingresar a sus usuarios a nuestro sistema.

Es el sitio indicado para que las empresas realicen actividades como:

- Compras
- Licitaciones y
- Subastas

de una manera rápida y confiable.

La sección de Compras maneja un amplio y variado catálogo de productos, ingresados por las empresas vendedoras, con los cuales los usuarios de cualquier empresa del sitio pueden generar sus órdenes de acuerdo a las necesidades de su empresa.

Eguana ofrece la oportunidad a las empresas del sitio de realizar licitaciones a otras empresas, en base a propuestas que incluyan las necesidades de quien licita. Eguana facilita este proceso y ayuda a las empresas ofertantes manteniéndolas informadas del flujo que lleve la licitación.

Los procesos de subastas también han sido incluidos dentro de Eguana debido a requerimientos de las empresas, permitiendo proponer un listado de productos a todas las empresas del sitio que deseen participar.

**Inicio**  
Compras Asignadas  
Licitación  
Subastas  
Salir de Eguana

**Promociones**

Almacenes Japón  
Ollas con garantía de exportación.

**Nuestras Empresas**

Existen diversas empresas que han se han decidido por Eguana a la hora de establecer sus relaciones comerciales con otras empresas, entre las que figuran empresas de:

- Electrodomésticos
- Papelería y Oficina
- Supermercados
- Librerías
- Música
- Hardware y software
- Telecomunicaciones
- Construcción, etc.

Entre las empresas que nos acompañan están:

- Comandato
- Alm. Japón
- Supermaxi
- Juan Marçet
- Librería Científica
- y muchas más.

Listo Intranet local

**Figura 37: Pantalla inicio de Eguana**

### 3.3 Integración con los demás módulos

La naturaleza modular de Eguana permite que sus módulos sean relativamente independientes los unos de los otros. El producto final debe de estar perfectamente integrado para poder ofrecer la funcionalidad completa de Eguana. La integración de los módulos independientes se la hace a través del modelo de Eguana, esto es los EJBs CMP y la base de datos. El modelo es el componente que todos los módulos tienen en común, todos usan la misma base de datos, todos usan los mismos EJBs CMP. Además otra parte de la integración se encuentra en la interfase gráfica, el módulo storefront debe de proveer enlaces a los demás módulos de administración, licitación y subasta, y reportes. A continuación una lista de los recursos que se comparten entre los módulos.

<b>Nombre del recurso</b>	<b>Módulos que lo usan</b>	<b>Módulo que lo implementa</b>
Base de datos	Todos	Todos

UsuarioBean	Todos	Administración
EmpresaBean	Todos	Administración
ProductoBean	Todos	Administración
CatalogoBean	Todos	Administración
RolBean	Todos	Administración
CompraBean	StoreFront, Licitación	StoreFront
DetalleCompraBean	StoreFront, Licitación	StoreFront
MensajeBean	StoreFront, Licitación	StoreFront
BancoBean	StoreFront, Licitación	StoreFront, Administración
CuentaBancoBean	StoreFront, Licitación	StoreFront, Administración
ProductorMensajesBean	StoreFront, Licitación	StoreFront, Administración

**Tabla 19: Recursos compartidos en Eguana**

En la implementación específica de JBOSS y J2EE la forma de compartir estos recursos es mediante un JAR<sup>1</sup> (Java Archive) común para todos los módulos. El nombre de este JAR es “eguanasuperjar.jar”. El nombre proviene del hecho que este JAR contiene todas las clases comunes de Eguana, independiente del módulo que provengan. Se ha escogido un estándar de empaquetamiento en este JAR. A continuación una breve descripción de este estándar:

---

<sup>1</sup> JAR (Java Archive). Archivo de java comprimido que contiene clases y recursos de java. Dependiendo del contexto de su uso puede ser un ejecutable de java, un repositorio de librerías e incluso una aplicación de JBOSS.

El nombre completo de las clases sigue el siguiente patrón:

```
org.eguana.{nombre del módulo que lo implementa | comun}.{wf | bo |  
mdb | clases | dto}.[ejb | proxies | utils]
```

comun: El paquete contiene clases simples de java usadas por todos los módulos.

wf: (Work Flow) El paquete contiene session Beans.

bo: (Bussiness Objects) El paquete contiene EJBs CMP.

mdb: (Message Driven Beans) El paquete contiene Message Driven Beans.

clases: El paquete contiene clases simples de java.

dto: El paquete contiene clases usadas por Sofia Framework.

ejb: Se lo usa después de wf,bo,mdb cuando el paquete contiene las clases principales de los EJBs.



proxies y utils: Se lo usa después de wf,bo,mdb cuando el paquete contiene clases secundarias o de ayuda de los EJBs.

Ejemplos:

La clase EJB CMP de Usuario:

org.eguana.administracion.bo.ejb.UsuarioBean

La clase EJB CMP de Empresa:

org.eguana.administracion.bo.ejb.EmpresaBean

La clase EJB CMP de Compra:

org.eguana.storefront.bo.ejb.CompraBean

La clase EJB Stateful Session OrdenCompraSession:

org.eguana.storefront.wf.ejb.OrdenCompraSessionBean

La clase Bean CatalogoW:

org.eguana.storefront.clases.CatalogoW

### Integración vía enlaces

Además de compartir objetos de negocios, en la interfase gráfica de Eguana se encuentra un segundo nivel de integración mediante simples enlaces hacia las interfaces de los demás módulos.

Desde el módulo StoreFront tenemos los siguientes enlaces hacia los demás módulos:

Sección	Enlace a módulo	Descripción
Ingreso al sistema	Administración	Antes de poder ingresar al sistema de Eguana, el usuario puede elegir entrar al sistema de administración de Eguana. No es necesario logonearse debido a que el sistema de administración tiene un sistema de login independiente.
Sección de compras y de revisión de ordenes	Licitación	El usuario una vez logoneado dentro del sistema puede entrar al módulo de licitación y subastas.
Sección de compras	Reportes	El usuario una vez

y de revisión de ordenes	logoneado dentro del sistema puede entrar el módulo de reportes.
--------------------------	--

**Tabla 20: Integración vía enlaces**

En conclusión la integración de todos los módulos en Eguana se consigue mediante tres vías: base de datos común, objetos de negocio comunes y enlaces entre módulos en la interfase gráfica.

## CONCLUSIONES

Una vez culminado el proyecto se pueden emitir las siguientes conclusiones:

Los modelos de arquitectura definen fundamentalmente las características principales de un sistema y determinan la manera de codificarlo. Por ello, es sumamente importante seleccionar aquel modelo que se ajuste mejor al sistema a desarrollar. En particular, el Modelo de Arquitectura de Software de n Capas, contempla la independencia entra las diversas capas, lo cual permite que se hagan cambios en cualquiera de estas sin afectar a las demás. Este modelo fomenta la modularidad y la reusabilidad de los elementos del sistema.

La aplicación puede ser dividida entre varios servidores físicos, distribuyendo diferentes capas en distintas ubicaciones.

El uso de una arquitectura J2EE permite que la solución sea escalable de acuerdo a las necesidades que se vayan planteando con la evolución y crecimiento del negocio. Adicionalmente esta misma arquitectura elegida posibilita que la aplicación sea portable e implementada sobre diferentes plataformas. Dada la importancia de permitir la expansión de la funcionalidad del sistema, la solución utiliza en todos sus módulos estándares que limitan el impacto de los cambios en el caso que sea necesario actualizar o crear nuevos componentes, reduciendo consecuentemente los costos y tiempos de desarrollo y mantenimiento.

UML también intenta solucionar el problema de propiedad de código que se da con los desarrolladores. Al implementar un lenguaje de modelamiento común para todos los desarrollos se

crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado para el desarrollo.

Eguana y específicamente el módulo StoreFront logró demostrar el uso del espectro completo de las tecnologías basadas en J2EE y de las herramientas y plataformas de código abierto. No solo se demostró la utilidad de la tecnología J2EE y del movimiento “Open Source”, también se puso en evidencia su poder y alcance en la industria de desarrollo de software para empresas.

Lamentablemente durante el desarrollo de este proyecto también se puso en evidencia las limitaciones de estas tecnologías y el porque aún no han logrado vencer completamente a su competencia.

La especificación J2EE 1.3 y 1.4, en especial la especificación EJB 2.x, es aún muy difícil de codificar y entender, incluso con la ayuda

de herramientas como XDoclet y Sofia. La curva de aprendizaje de esta tecnología y su tiempo de desarrollo es la barrera más grande que existe para su total aceptación en el mercado. Tecnologías rivales como .Net de Microsoft sostienen todavía una ventaja en este aspecto.

Como se pudo apreciar en la secciones de Catálogo, Proceso de compra, despacho y cancelación en el capítulo 3, existían circunstancias donde era contraproducente usar los EJBs. Estos eran demasiado ineficientes para ciertos tipos de aplicaciones y escenarios.

El soporte al desarrollador y al usuario para las tecnologías J2EE y herramientas código abierto todavía necesitan madurar. En algunos casos los sistemas de ayuda y soporte eran inexistentes o la información se encontraba desorganizada e inexacta, incluso cuando se pagaba por el acceso a la misma.

Afortunadamente la comunidad de Java ha tomado grandes pasos para remediar estos problemas. Eguana fue desarrollado con el estándar J2EE 1.3, el cuál aplica la especificación EJB 2.0. La última versión en producción de JBOSS, la 4.0, implementa el estándar J2EE 1.4 con EJB 2.1. Actualmente existe en estado de revisión pública el estándar J2EE 5<sup>1</sup> con EJB 3.0.

EJB 3.0 simplifica de una manera dramática la implementación de los EJBs. En esta nueva especificación se elimina, entre muchas cosas, el uso de las interfaces remotas, locales, y objetos “Home” y la necesidad del complicado descriptor de despliegue. Los EJBs de entidad son más parecidos a los POJOs (Plain Old Java Objects) y a los Java Beans. A pesar de no estar finalizada, esta especificación es implementada por JBOSS 4.0 de manera experimental. Con la ayuda de Hibernate, EJB 3.0 y J2EE 5 se posicionan fuertemente en la industria de desarrollo de

---

<sup>1</sup> J2EE 5 es la versión siguiente a J2EE 1.4. La comunidad de Java, por simplificación, decidió no usar el número 1.5 en las últimas versiones de J2EE y Java SDK.



software empresarial, ofreciendo un producto robusto, completo y simple a la comunidad de desarrolladores.

## RECOMENDACIONES

En el caso que Eguana sea implementado y usado por alguna empresa en el futuro se recomienda lo siguiente:

Modificar el diseño del sistema para las nuevas versiones:

Java SDK 5

J2EE 5

EJB 3.0

JBOSS 4.0

MySQL 5.0

Añadir un mayor nivel de comunicación entre el vendedor y el comprador como por ejemplo un servicio de "Chat" dentro del StoreFront.

Aumentar las opciones de pago disponibles para el comprador, como por ejemplo el uso de tarjetas de crédito.

Añadir un servicio de rastreo de paquetes despachados para que el usuario pueda conocer el estado y localización de su paquete mientras éste llega a su destino.

Desarrollar una interfase amigable para dispositivos móviles como celulares y PDAs<sup>1</sup>.

---

<sup>1</sup> PDA: De sus siglas en inglés Personal Digital Assistant

## **GLOSARIO DE TÉRMINOS**

### **Aplicación J2EE**

Cualquier unidad desplegable con funcionalidad J2EE. Este puede ser un simple módulo J2EE o un grupo de módulos empaquetados dentro de un archivo EAR con un descriptor de despliegue de la aplicación J2EE.

### **Aplicación Web**

Una aplicación escrita para internet, incluyendo tecnología Java como JSP y Servlets o tecnologías no Java como CGI.

### **Base de Datos**

Conjunto de datos no redundantes, almacenados en un soporte informático, organizados de forma independiente de su utilización y accesibles simultáneamente por distintos usuarios y aplicaciones. Los

datos se almacenan cumpliendo tres requisitos básicos: no redundancia, independencia y concurrencia.

### **Browser**

Navegador de Internet. (Netscape, Explorer)

### **Cliente/Servidor**

Arquitectura de sistemas de información en la que los procesos de una aplicación se dividen en componentes que se pueden ejecutar en máquinas diferentes. Modo de funcionamiento de una aplicación en la que se diferencian dos tipos de procesos y su soporte se asigna a plataformas diferentes.

### **Contenedor**

Es una entidad que provee administración del ciclo de vida, seguridad, despliegue y servicios de tiempo de ejecución a los componentes J2EE. Cada tipo de contenedor (EJB, Web, Applet y Application Client) también provee servicios de componentes específicos.

### **Contenedor EJB**

Es un contenedor que implementa los componentes EJB, en base a la arquitectura J2EE. La arquitectura especifica un ambiente de tiempo de ejecución para Enterprise Beans que incluye seguridad, concurrencia, ciclo de vida, administración, transacciones, despliegue, nombramiento y otros servicios. Un Contenedor EJB es provisto por un Servidor EJB o J2EE.

### **Deployment (Despliegue)**

Es el proceso mediante el cual el software es instalado dentro de un ambiente operacional.

### **Deployment descriptor**

Es un archivo XML proveído por cada módulo y aplicación J2EE que describe como ellos van a ser desplegados.

### **EAR Archivo**

Enterprise Archivo File. Un archive JAR que contiene una aplicación J2EE.

### **Enterprise Bean**

Es un componente J2EE que implementa una tarea del negocio o una entidad del negocio y esta hospedada por un contenedor EJB, puede ser un Entity Bean, un Session Bean, o un Message-Driven Bean.

### **Enterprise JavaBeans (EJB)**

Es una arquitectura de componentes para el desarrollo y despliegue de aplicaciones orientada a objetos, distribuidas y de nivel empresarial. Las aplicaciones escritas usando la arquitectura Enterprise JavaBeans son escalables, transaccionales y seguras.

### **Entity Bean**

Es un Enterprise Bean que representa la persistencia de datos mantenidos en una base de datos. Un Entity Bean puede manejar su propia persistencia o puede delegar esta función a su contenedor. Un Entity Bean es identificado por una clave primaria. Si el contenedor en el cual el Entity Bean es hospedado se cae, el Entity Bean, su clave primaria, y cualquier referencia remota sobrevive a la caída.

### **Etiqueta (Tag)**

En los documentos XML, una pieza de texto que describe una unidad de datos o un elemento.



## **HTML**

Hypertext Markup Language. Lenguaje para escribir documentos en Internet. HTML habilita el uso embebido de imágenes, sonido, flujo de video, formularios, referencias de URLs y formateo básico de texto.

## **HTTP**

Hypertext Transfer Protocol. Es el Protocolo de Internet usado para obtener objetos hipertexto desde hosts remotos. Los mensajes HTTP consisten en requerimientos desde clientes hacia servidores y respuesta desde servidores a clientes.

## **HTTPS**

HTTP sobre el protocolo SSL.

## **Internet**

Se refiere a la red más grande del mundo, conecta miles de redes con alcance mundial.

## **JAR**

Archivo Java. Un formato de archivo de plataforma independiente que permite que muchos archivos sean agregados dentro de un archivo final.

## **Java 2 Platform, Enterprise Edition (J2EE)**

Es un ambiente para desarrollo y despliegue de aplicaciones empresariales. La plataforma J2EE consiste de un conjunto de servicios, APIs y protocolos que proveen la funcionalidad para desarrollo multi hilo y aplicaciones basadas en Web.

## **Java Naming and Directory Interface (JNDI)**

Es una API que provee funcionalidad de nombramiento y directorios.

## **JavaBeans component**

Es una clase de Java que puede ser manipulada por herramientas y compuesto dentro de aplicaciones. Un componente JavaBeans puede adherirse a ciertas propiedades y eventos de las convenciones de la Interface.

### **JavaMail**

Es un API para envío y recepción de email.

### **JavaServer Pages (JSP)**

Es una tecnología Web extensible que usa datos estáticos, elementos JSP, y objetos Java de lado del servidor para generar contenido dinámico en clientes. Típicamente los datos estáticos son elementos HTML o XML, y en muchos casos el cliente es un Navegador o Browser.

### **JDBC**

Es un API para conectividad, independiente de la base de datos, entre la plataforma J2EE y un amplio rango de Data Sources (Fuente de Datos).

### **Lógica del Negocio**

Es el código que implementa la funcionalidad de una aplicación.

### **Mensaje**

En JMS, una petición, reporte o evento asincrónico que es creado, enviado y consumido por una aplicación empresarial y no por un ser humano. Este contiene información vital necesaria para coordinar aplicaciones empresariales, en la forma de datos formateados con precisión que describen acciones del negocio específicas.

### **Método del Negocio**

Es un método que implementa una lógica del negocio o las reglas de la aplicación.

### **Módulo J2EE**

Una unidad de software que consiste de uno o más componentes J2EE del mismo tipo de contenedor y un descriptor de despliegue de ese tipo.

### **Plantilla (Template)**

Un conjunto de instrucciones con formato que aplican a los nodos seleccionados de una expresión Xpath.

### **Persistencia manejada por el Contenedor**

Es el mecanismo mediante el cual la transferencia de datos entre las variables y el manejador de recursos de un Entity Bean es manejado por el contenedor del Bean.

### **Sistema de Mensajería Punto a Punto**

Este sistema es construido bajo el concepto de colas de mensajes.

Cada mensaje es direccionado a un cola específica; los clientes extraen mensajes de las colas establecidas para cargar esos mensajes.

### **Servlet**

Programa en Java que se ejecuta como parte de un servicio de red, típicamente un servidor del HTTP y responde a las peticiones de clientes.

### **SQL (Structured Query Language)**

Lenguaje de interrogación normalizado para base de datos relacionales.

SQL es un lenguaje de alto nivel, no procedural, normalizado, que permite la consulta y actualización de los datos de BD relacionales. Se

ha convertido en el estándar para acceder a base de datos relacionales.

El SQL facilita un lenguaje de definición de datos y un lenguaje de

manipulación de datos. Además incluye una interfase que permite el acceso y manipulación de la BD a usuarios finales.

### **SSL Secure Socket Layer**

Es un protocolo para transmitir nuestra información a través del Internet de manera encriptada.

### **UML**

Es un lenguaje gráfico que sirve para modelar, diseñar, estructurar, visualizar, especificar, construir y documentar software.

### **Transacción**

Una unidad atómica de trabajo que modifica datos.

### **URL**

Uniform resource locator. Un estándar para escribir una referencia textual a una pieza arbitraria de datos en el Web.

## **Virtual Private Network**

Es una red privada que se extiende en base a un proceso de encapsulación y encriptación de los paquetes de datos a distintos puntos remotos mediante el uso de unas infraestructuras públicas de transporte

## **WAR**

Web Application Archive File. Un archive JAR que contiene un módulo Web.



## **ANEXO 1**

### **Descripción de la Aplicación Bancaria**

Dentro del proceso de compras, el siguiente paso a la generación de la orden es el pago electrónico del monto al proveedor. Para realizar este proceso, se creó un módulo o aplicación que simula un sistema bancario, que actúa como una institución financiera que mantiene toda la información referente a los pagos de valores monetarios involucrados en las transacciones de compras. Esta aplicación trabaja con las cuentas de las empresas compradoras y vendedoras para realizar las transacciones de débito o crédito correspondientes.

El pago electrónico se realiza por orden generada. Esto se debe a que si los productos adquiridos por la empresa compradora pertenecen a n empresas vendedoras, se generan n órdenes de compra para poder efectuar el pago a cada empresa implicada.

En esta aplicación bancaria se recibe peticiones para la aprobación de transacciones generadas en Eguana. Una vez recibida la petición mediante un mensaje asíncrono se genera en el sistema una aprobación en espera. Realizada la verificación por parte de un banquero ficticio o simulado, la aplicación bancaria envía un mensaje al módulo Storefront el cual procede al cambio de estado de la orden de compra, dependiendo del resultado. Además en la simulación bancaria se realizan los débitos de la cuenta de las empresas compradoras y se acreditan a las empresas vendedoras el valor de la orden correspondiente, y ésta pasa al estado Pagada en Eguana. De no ser aceptada la aprobación en espera, la orden pasa a estado No Aprobada y se da por terminado el proceso de compra de dicha orden.

Esta comunicación se lleva a cabo gracias a la existencia de una aplicación de mensajería, descrita en un capítulo anterior, para la cual se crearon clases que permiten realizar la comunicación con el módulo Storefront.

La implementación de esta aplicación bancaria consta de los siguientes objetos:

<b>Tabla</b>	<b>Descripción</b>
CUENTA	Tabla que mantiene los datos de las cuentas de las empresas del sistema.

**Tabla 21: Estructuras de base de datos**

<b>Clase</b>	<b>Tipo</b>	<b>Descripción</b>
BancoSimBean.java	Bean de Sesión (Session Bean) sin estado	Contiene el método que realiza las operaciones de

		crédito/débito en las cuentas de las empresas implicadas en la compra.
CuentaBean	Bean de Entidad (Entity Bean)	Bean que hace referencia a la tabla CUENTA conteniendo los métodos correspondientes.
MensajesABancoBean	Bean manejado por mensajes (Message-Driven Bean)	Se encarga de ejecutar la transacción de débito/crédito y se comunica con Storefront a través de mensajería.
ProductorMensajes	Clase Java	Se encarga del envío de mensajes desde el módulo bancario hacia Storefront.
AprobacionBean	Bean de Entidad (Entity Bean)	Bean que hace referencia a la tabla APROBACION conteniendo los métodos correspondientes.

**Tabla 22: Objetos J2EE de la aplicación bancaria**

Al igual que las clases del módulo Storefront, estas clases mantienen código XDoclet para la generación de configuración en los descriptores XML del servidor JBoss. Se usó la tecnología “DBFORMS” para la interfase gráfica de la aplicación.

## BIBLIOGRAFÍA

1. David de la Fuente García, Nazario García Fernández, Isabel Fernández Quesada, mayo 2003, E-Procurement: Importancia y Aplicación, Escuela Técnica Superior de Ingenieros Universidad de Oviedo, Asturias España.
2. Graham Hayday, abril 2003, U.K. government signs Linux deal, <http://news.com.com/2100-1012-995544.html>
3. Eric Armstrong, Jennifer Ball, Stephanie Bodof, Debbie Bode Carson, Ian Evans, Dale Green, Kim Haase, Eric Jendrock, The J2EE™ 1.4 Tutorial, junio 17, 2004.
4. Fernando Rodríguez, junio 2004, J2EE y EJB, [www.fing.edu.uy/inco/cursos/tsi1/2004/teorico/Plataforma J2EE y EJBs 2004.ppt](http://www.fing.edu.uy/inco/cursos/tsi1/2004/teorico/Plataforma_J2EE_y_EJBs_2004.ppt)
5. JC Dueñas, abril 2003, Modelo-Vista-Controlador, <http://polaris.dit.upm.es/~jcduenas/patrones/Modelo.htm>
6. CodeJava, abril 2005, Struts y el patrón de diseño MVC (model-view-controller), <http://www.codejava.org/?idxpagina=9&idxnota=32795&destacada=1>
7. Sesión 1: Introducción a la Tecnología EJB, abril 2005, <http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/ejb/sesion01-apuntes.htm>
8. Wikipedia, octubre 2005, Linux, <http://en.wikipedia.org/wiki/Linux>

9. JBoss, octubre 2005, JBoss Application Server, <http://www.jboss.com/products/jbossas>
10. JBoss, diciembre 2004, JBoss Application Sever 4, <http://www.jboss.com/elqNow/elqRedir.htm?ref=/pdf/JBossASBrochure-Jun2005.pdf>
11. Scott Stark y The JBoss Group, JBoss Administration and Development (3ra edición junio 2003)
12. MySQL, octubre 2005, MySQL Pro Certified Server, <http://www.mysql.com/products/database/mysql/>
13. Eclipse.org, octubre 2005, Eclipse.org, <http://www.eclipse.org/>
14. Salmon LLC, octubre 2005, Sofia JSP GUI Java Development Framework, <http://www.salmonllc.com/website/Jsp/vanity/Sofia.jsp>
15. Sun Microsystems, octubre 2005, Java Blueprints – J2EE Patterns, <http://java.sun.com/blueprints/patterns/MVC-detailed.html>
16. The Server Side, marzo 2005, Application Sever Matrix, <http://www.theserverside.com/reviews/matrix.tss>
17. Universidad de Chile, febrero 2005, <http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html>.
18. Rodrigo Guridi, Claudia Rostagnol, Gerardo Canedo, 2004, Informe Proyecto: Gestión de Bookmarks, <http://www.fing.edu.uy/inco/cursos/tsi/TSI3/2004-trabajos/TSI3-2004-Gr1A-Bookmarks.pdf>
19. Programación.com, marzo 2005, [http://www.programacion.com/java/tutorial/jap\\_aplic\\_jboss/2/](http://www.programacion.com/java/tutorial/jap_aplic_jboss/2/)

20. Sun Microsystems, Inc. , agosto 1998, JavaMail™ Guide for Service Providers.
21. Marty Hall, marzo 2002, Declarative Web Application Security with servlets and JSP,  
<http://www.informit.com/articles/article.asp?p=26139&redir=1>
22. JBoss.org, junio 2005, Chapter 8. Security on Jboss,  
<http://docs.jboss.org/jbossas/jboss4guide/r1/html/ch8.chapter.html>
23. Natividad Martínez, Parte V: Nivel de presentación Servlets y JSPs (Madrid, Universidad Carlos III de Madrid, 2003)