Desarrollo de una tienda virtual en una arquitectura distribuída

Gustavo .A. Javier Garrido
ETIS

Josep Maria Camps Riba

18/6/2004

Dedicatoria y agradecimientos

Es bien sabido que el estudio de una nueva tecnología conlleva éxitos y frustraciones a lo largo del período de aprendizaje. Estos éxitos y frustraciones se traducen en constantes cambios de humor en la persona que acomente el reto.
Dedico lo que ha significado este trabajo para mí, en tiempo y esfuezo, a las personas que lo han soportado y comprendido, en especial mi familia.

RESUMEN
Desarrollo de una tienda virtual en una arquitectura distribuida.

Una *tienda virtual* es una subcategoría de una familia más grande de aplicaciones llamadas sistemas de comercio electrónico.
Un sistema de comercio electrónico posibilita que compradores y proveedores de productos y servicios realicen transacciones business-to-business o business-to-consumer en un mercado virtual.
Los tres conceptos que subrayan una tienda virtual son: escaparate (listado de productos categorizados), cesta y pedido. En concreto, una tienda virtual, debe permitir al usuario escoger los productos que desee, separándolos en lo que se llama una cesta, debe permitir modificar y eliminar productos de esta cesta y finalmente permitir la concreción de la transacción.
Para conseguirlo, se utilizará una *arquitectura multicapa o distribuida*, esto es, la división de la responsabilidades del sistema en bloques semánticos. En concreto, estas capas serán las siguientes: presentación, negocio y datos.
La buena distribución de las capas permitirá conseguir los objetivos de esta arquitectura, esto es: inteligibilidad del código, fácil mantenimiento, alto nivel de producción, reutilización y portabilidad.
Las capas pueden estar alojadas en la misma máquina virtual o en diversas, en cualquier caso, esto es transparente para el cliente.
Como contraposición, en el desarrollo de la parte gestora no se utiliza dicha arquitectura, sino una más simple, que elimina la posibilidad de un servidor intermedio entre la capa de presentación y de negocio. La finalidad de dicho gestor es la de actualizar los productos y categorías disponibles desplegados en la tienda.
A lo largo del presente trabajo, se presentan las tecnologías y procesos que se han usado y seguido en la construcción de dicha tienda virtual y su gestor. El objetivo, es la creación de un software flexible, portable, reutilizable, con alto rendimiento e inteligible.
En concreto la *plataforma* de desarrollo será J2EE, en ella se implementarán los componentes correspondientes (EJBs, JSPs, Servlets..).
El *proceso* seguirá las directrices RUP y los diagramas UML.

Para llevar a cabo el proyecto se utiliza el servidor de aplicaciones *Jboss*, junto con el servidor web *Tomcat* y el sistema gestor de base de datos *Mysql*.

Índice de contenidos

Índice de figuras

CAPÍTULO 1

INTRODUCCIÓN

A nivel práctico el proyecto consta de dos partes, el desarrollo de una tienda virtual con funcionalidades mínimas y un gestor de contenidos para dicha tienda. La tienda estará organizada por categorías y permitirá realizar búsquedas de productos, almacenar una selección de los mismos en una cesta y finalmente hacer el pedido. El gestor se encargará de actualizar dichos productos.

Un proyecto mediano de programación requiere una serie de pasos previos a la implementación con el objetivo de conseguir una correcta interpretación de las necesidades del mismo, reaprovechar software existente y posteriormente permitir un mantenimiento fácil.
El Rational Unified Process, RUP, es un framework o estructura para el proceso de desarrollo de software. Mediante una serie de pasos (disciplines) asigna tareas y responsabilidades en una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad, que satisfaga a los usuarios, dentro de un tiempo y presupuesto predecible, *"better software faster"*.

Tecnológicamente se emplearán EJBs bajo la plataforma J2EE.
La arquitectura de componentes acoplados ligeramente (loosely coupled components) ofrece flexibilidad, escalabilidad y portabilidad.
La arquitectura EJB es una especificación desarrollada por Sun Microsystems. Describe una arquitectura basada en componentes que facilita el desarrollo e instalación de *aplicaciones distribuidas*.

La especificación detalla los servicios y requisitos de un servidor de aplicaciones que gestiona componentes EJB. También describe los requisitos de código que los desarrolladores de beans deben seguir para crear aplicaciones portables. La meta más importante es que los desarrolladores de beans programen componentes EJB una vez y puedan instalarlos en cualquier servidor de aplicaciones que cumpla con la tecnología EJB. Además, la arquitectura EJB consigue que las aplicaciones de empresa sean escalables, seguras y transaccionales.

Las aplicaciones distribuidas requieren acceso a una serie de servicios de empresa. Los servicios t¡picos incluyen procesamiento de transacciones, acceso a bases de datos, mensajería, multihilos (multithread),etc. La arquitectura de J2EE unifica el acceso a estos servicios en un API de servicios de empresa. Sin embargo, en lugar de tener que acceder a estos servicios a través de interfaces de propietario o no estándar los programas de aplicación en J2EE pueden acceder a estos API mediante el contenedor.

Una típica plataforma J2EE comercial (o servidor de aplicación J2EE) incluye uno o más contenedores y el acceso a los API de empresa viene especificado por J2EE.

El presente trabajo tiene como finalidad ilustrar los pasos, decisiones y cuestiones de implementación que se han tomado para la construcción de la 'tienda virtual' en una arquitectura distribuida, siguiendo el proceso RUP y razonando la eficiencia de la implementación en la arquitectura EJB. Esto último es tan importante como lo primero debido a que los recursos consumidos por un servidor J2EE pueden resultar caros.

Se ha usado para la implementación el servidor de aplicaciones JBoss, el servidor de Servlets Tomcat y el gestor de base de datos MySql.

JUSTIFICACIÓN

La ingeniería de software comprende los métodos y técnicas que se utilizan en el desarrollo profesional de software. Consta de dos familias, la estructurada y la orientada a objetos. Esta última, que es la utilizada en el presente trabajo, consta de un modelo (UML), una técnica (que define el ciclo de vida de la aplicación, RUP en este caso) y un lenguaje (orientado a objetos, JAVA en este caso).

La arquitectura de software engloba el conjunto de decisiones respecto a la organización de un sistema de software.

El presente trabajo combina los tres aspectos esenciales en el aprendizaje del desarrollo profesional de software:

Tecnología: UML, JAVA, J2EE
Proceso: The Rational Unified Process (RUP)
Ejemplo: Tienda virtual y Gestor de contenidos

Tienda virtual

La razón de desarrollar una tienda virtual es que presenta toda una gama de interacciones entre componentes y cliente-servidor que se pueden trasladar fácilmente a cualquier otro tipo de e-comercio. Lo importante pues son los conceptos que subyacen a la implementación.

Sistemas distribuidos

Desde hace un tiempo que se conoce que la mejor solución en el desarrollo de sistemas es dividir las responsabilidades en diversas capas, lo que da como resultado una arquitectura multicapa o distribuida. Así, en una aplicación de tres capas, por ejemplo, tendremos la capa de presentación responsable de atender las interacciones con el usuario, la capa de negocio responsable de las reglas de negocio (condiciones que se deben cumplir para la ejecución de un proceso) y la capa de integración que provee acceso a los datos necesarios para la aplicación. Esta división permite que el contenido de cada capa sea cambiado independientemente.

Plataforma J2EE

Los desarrolladores de beans se pueden concentrar en el diseño de aplicaciones de empresa, los requerimientos de las reglas de negocio y los procesos. No es necesario escribir código para las transacciones de bases de datos, por ejemplo. A medida que la calidad de los servidores de aplicaciones aumente, la misma aplicación escrita anteriormente mejorará su rendimiento sin cambiar nada en su código.

Esto significa que se pueden escribir aplicaciones con transacciones, multiusuario y escalables sin ser expertos en transacciones, programación multihilo, seguridad o programación de base de datos. Esto no quiere decir que programar EJBs sea simple, pero es accesible y portable.

La plataforma J2EE ayuda a superar varios de los problemas que planteaba la informática de empresa.

RUP

Un proceso de desarrollo de software efectivo debe describir quién hace qué, cómo y cuando. RUP implementa exactamente esto en los términos de los siguientes conceptos clave:

- Roles (actores o responsables): Quién
- Artifacts (finalidad de una fase): Qué
- Activities (pasos en una fase): Cómo
- Fases, iteraciones, disciplinas y detalles de flujo: Cuando

UML

El *Unified Modeling Language* es un lenguaje gráfico para el modelaje y desarrollo de sistemas de software. Provee la infraestructura para la visualización y el modelaje de todas las fases en el desarrollo de software, desde los requerimientos a la especificación, la construcción y el despliegue.

La idea central al usar UML es capturar los detalles más significativos del sistema, de manera que el problema sea claramente entendido, y se identifique y construya una arquitectura y una implementación.

TFC

A lo largo de la carrera de 'Enginyieria Tècnica de Sistemes' se han visto muchas de las aplicaciones de Java en el mundo de la programación. Elegir este TFC no sólo significa llegar a la *cima* en cuanto a lo que ofrece Java en la informática profesional sino invertir en unos conocimientos que conforman el presente y futuro de la informática empresarial.

OBJETIVOS DEL TFC

El propósito de este trabajo es obtener una visión clara de la plataforma J2EE, el desarrollo de software por componentes, la informática distribuida, los procesos de desarrollo de software y el manejo de UML respecto a Java.

El punto de partida es el resumen de las funcionalidades mínimas de la aplicación a implementar, en este caso, una tienda virtual y un gestor de contenidos para dicha tienda.
La aportación del TFC radica en el seguimiento e ilustración del proceso seguido y las tecnologías usadas.
El contexto en el cual se desarrolla es el de las aplicaciones multicapa.

Se ha considerado que la implementación no sea simplemente un producto funcional, sino que también sea eficiente. Esto es tan importante como lo primero, dado que los recursos del sistema que emplea un servidor de aplicaciones pueden resultar muy caros en cuanto a rendimiento y sobrecarga. Por otra parte se ha cuidado la prolija implementación de los diferentes componentes y patrones, de manera de conseguir varios de los objetivos de la informática de empresa: reusabilidad, fácil actualización y portabilidad.

En concreto, los objetivos se pueden ver en los puntos siguientes:

1. Seguimiento de un proceso de construcción de software (RUP)
2. Entender y describir los requerimientos de la aplicación usando diagramas UML.
3. Explorar la relación entre Java y UML.
4. Uso de java para crear aplicaciones del lado del servidor con JavaServer Pages, Servlets, Javabeans y EJBs.
5. Implementación de una aplicación con componentes distribuidos.
6. Uso de patrones.
7. Eficiencia, rendimiento y portabilidad en la implementación
8. Familiarización con los métodos de seguridad en el acceso a aplicaciones.

9. Investigar los beneficios de desplegar aplicaciones Java en productos de código abierto.

Su traslación a la práctica en una aplicación web funcional utiliza el servidor de aplicaciones JBoss, el servidor web Tomcat y el servidor de base de datos Mysql.

Los capítulos que siguen describen los pasos seguidos en el desarrollo de dicha tienda virtual, las decisiones respecto a su arquitectura y el método seguido.

ENFOQUE Y MÉTODO SEGUIDO

Se ha seguido el enfoque RUP (parcialmente implementado, sólo cuatro etapas, y ligeramente diferente, etapa de análisis y diseño por separado), o sea, se ha dividido el desarrollo de la aplicación en las siguientes cuatro etapas o disciplinas: requisitos, análisis, diseño e implementación. En la elaboración de los diagramas correspondientes a cada etapa se ha utilizado UML.
Respecto a la arquitectura, la tienda virtual se implementa en una arquitectura multicapa basada en ejb's, mientras que el gestor utiliza javabeans para implementar las condiciones del proceso (capa de negocio). A lo largo de las etapas se descubren los diferentes componentes software que se han de implementar y sus relaciones.

Rup es un arquitectura de software iterativa e incremental. Se basa en una serie de pasos que conducen principalmente a la implementación de un software que cumple las especificaciones exigidas, a una documentación que facilita el mantenimiento y a un aprovechamiento de sotware ya probado (componentes) y de métodos también ya probados (patterns). Hoy en día el software no sólo debe funcionar sino que también debe permitir ser leído y entendido correctamente. El software que no cumple este requisito se entiende que es un software de calidad pobre, el cual es difícil de mantener, de reaprovechar y , posiblemente, hasta serà redundante, compromentiendo el rendimiento y la seguridad.

Principalmente estos pasos lo conforman una serie de dibujos (diagramas UML) que facilitan la comprensión visual, mostrando las relaciones e interacciones de las diferentes 'piezas' que componen el software.

No es un método rígido, ni obligatorio, por ejemplo en una aplicación sencilla sería contraproducente sumergirse en el proceso Rup, por otro lado se disponen de una serie de diagramas que es el ingeniero quien debe

decidir usar, teniendo en cuenta que lo importante es el objetivo de cada fase y su comprensión.

Para el desarrollo de la 'tienda virtual' se han seguido las cuatro primeras fases del modelo Rup, en cada una de ellas se han empleado las técnicas diagramáticas y textuales que se han considerado más adecuadas para conseguir el objetivo de cada fase:

Requisitos: Llegar a una especificación del software a desarrollar, esto es entender con precisión cual debe ser la funcionalidad del software:

Enunciado
Modelo de dominio (entorno sobre el cual gira el software, clases básicas)
Especificación
Casos de uso
Guiones
Glosario
Documentación textual de los casos de uso

Análisis: Abstraer las clases esenciales y sus relaciones estáticas, las interficies y exponer el funcionamiento interno de los casos de uso más complejos así como realizar divisiones (paquetes) en cuanto a funcionalidad:

Paquetes de análisis
Especificación de las clases de análisis
Identificación de las clases frontera, control y de las operaciones
Diagramas de colaboración simplificados.
Diagramas de secuencia
Análisis de la interficie de usuario

Diseño: La finalidad del diseño es la de servir de puente entre el análisis y la implementación. En esta etapa ya se presentan resultados concretos de la apariencia interna y externa del software traducidas al lenguaje y las herramientas de desarrollo:

Diseño Frontera
Diseño Control
Diseño Entidad
Patterns

Implementación: En el capítulo de implementación se describen los conceptos claves que se han seguido el la programación de la aplicación. También se describen los componentes implementados

PLANIFICACIÓN DEL PROYECTO

Requisitos				
Análisis				
Diseño				
Implementación de la tienda				
Implementación del gestor				
Manual de instalación				
Memoria				
Presentación				
Fecha límite	8-3-2004	29-3-2004	19-4-2004	18-6-2004

Las etapas no acaban definitivamente en la fecha indicada, sino que evolucionan de acuerdo con el modelo iterativo e incremental utilizado. Teniendo en cuenta esto, los objetivos de cada etapa se han conseguido en las fechas previstas.

PRODUCTOS OBTENIDOS

Los productos obtenidos son los siguientes:

Documentación de los requisitos, análisis y diseño.
Estos puntos describen el proceso de desarrollo de la aplicación.

Implementación de la tienda virtual.
Código fuente y compilado de la implementación de la tienda virtual.

Implementación del gestor de contenidos.
Código fuente y compilado de la implementación del gestor.

Manual de instalación.
Pasos para la instalación del entorno y de la aplicación.

Fichero Build.xml
Compila e instala la aplicación en el servidor JBoss. Se ejecuta con la herramienta ANT.

Fichero BDLibreria.sql
Crea la base de datos usada por la aplicación e inserta unos registros iniciales. Para uso con MySql.

Memoria.
Repaso a la plataforma J2EE, descripción del proyecto, descripción del proceso de desarrollo y claves de la implementación.

Presentación de la aplicación en Powerpoint.
Síntesis del proceso realizado en la implementación de la aplicación.

DESCRIPCIÓN DEL RESTO DE CAPÍTULOS

En el resto de capítulos se puede encontrar lo siguiente:

Capítulo 2: *Tecnología.*
Breve repaso de la plataforma J2EE.

Capítulo 3: *Recogida de requisitos.*
Primer paso en la formalización y especificación del sistema.

Capítulo 4: *Análisis*.
Consecusión de las primeras clases del sistema y sus relaciones.

Capítulo 5: *Diseño*.
Traslación al lenguaje y plataforma utilizados. Diseño de la interfaz. Patrones. Modelo de datos.

Capítulo 6: *Implementación*.
Detalles de los componentes y clases implementadas, así como de los patrones escogidos.

Capítulo 7: *Conclusión*.
Breve valoración de las dificultades y facilidades en el desarrollo de esta arquitectura. Mejoras a realizar.

CAPÍTULO 2

TECNOLOGÍA

La plataforma J2EE

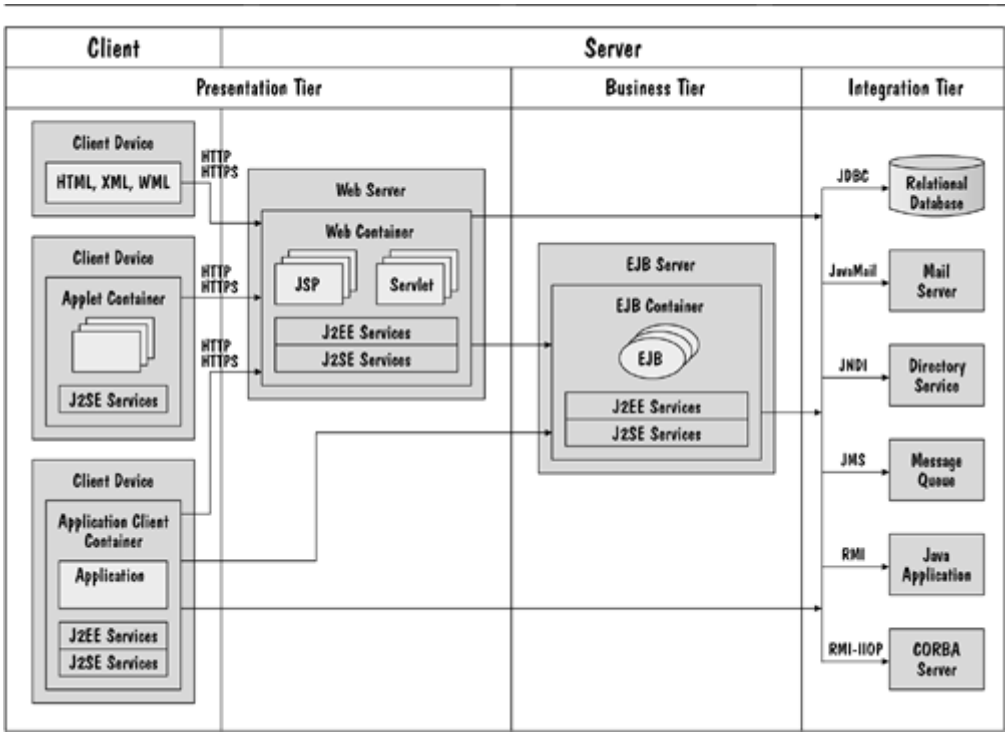
Enterprise JavaBeans

La informática distribuida siempre ha tenido problemas que resolver: seguridad, concurrencia, transacciones en las bases de datos, integridad de los datos, y requisitos de rendimiento son sólo algunos de ellos. La plataforma J2EE ha sido creada para darles solución, mediante la arquitectura EJB.

La arquitectura de componentes acoplados ligeramente (loosely coupled components) ofrece flexibilidad, escalabilidad y portabilidad.
La arquitectura EJB es una especificación desarrollada por Sun Microsystems. Describe una arquitectura basada en componentes que facilita el desarrollo e instalación de aplicaciones distribuidas.
La especificación detalla los servicios y requisitos de un servidor de aplicaciones que gestiona componentes EJB. También describe los requisitos de código que los desarrolladores de beans deben seguir para crear aplicaciones portables. La meta más importante es que los desarrolladores de beans programen componentes EJB una vez y puedan instalarlos en cualquier servidor de aplicaciones que cumpla con la tecnología EJB. Además, la.arquitectura EJB consigue que las aplicaciones de empresa sean escalables, seguras y transaccionales.

Los EJBs son componentes que se ejecutan dentro de un “contenedor EJB” bajo la supervisión del servidor de aplicaciones. Hay tres tipos de EJB: sesión, entidad y controlados por mensaje. Respecto a los de sesión, hay de de dos clases: sin estado y con estado. También hay dos clases de beans de entidad, aquellos cuya persistencia la controla el servidor y los que la persistencia la controla el bean.
El servidor de aplicaciones y el contenedor EJB proveen los servicios del sistema para los EJBs, tales como persistencia de datos, transacciones, seguridad, y administración de recursos. El contenedor EJB gestiona reservas de conexiones a la base de datos, como también reservas de instancias EJB que pueden ser asignadas a clientes cuando sea necesario.

La plataforma Java 2, Enterprise Edition (J2EE), es un conjunto estándar de APIs de Java proporcionada por Sun Microsystems. Incluye la arquitectura EJB y un conjunto de paquetes relacionados que logran la cohesión de la plataforma. Por ejemplo, un cliente Java puede usar los servicios de directorio (JNDI) para buscar la ubicación de un componente EJB. El servidor de aplicaciones, que provee de servicios de sistema para que funcionen los EJBs, usan Remote Method Invocation (RMI) y RMI-IIOP para hacer llamadas remotas a través de la red. Los beans controlados por mensaje usan el servicio de mensajes de Java (Java Message Service, JMS) que lo capacita para responder a mensajes. Así, mientras la tecnología EJB provee servicios específicos en el dominio de la informática de empresa, es parte de una imagen mucho mayor, que la proporciona J2EE y los muchos paquetes independientes que proveen servicios específicos.



El beneficio de todo esto es que los desarrolladores de beans se pueden concentrar en el diseño de aplicaciones de empresa, los requerimientos de las reglas de negocio y los procesos. No es necesario escribir código para las transacciones de bases de datos, por ejemplo. A medida que la calidad de los servidores de aplicaciones aumente, la misma aplicación escrita anteriormente mejorará su rendimiento sin cambiar nada en su código.

Ahora, los desarrolladores de beans, pueden escribir aplicaciones con transacciones, multiusuario y escalables sin ser expertos en transacciones, programación multihilo, seguridad o programación de base de datos. Esto no quiere decir que programar EJBs sea simple, pero es accesible y portable.

Sistemas distribuidos

Uno de los objetivos principales de la plataforma J2EE es proveer una infraestructura estándar para el desarrollo de sistemas o aplicaciones de empresa.
Una empresa es una organización económica y las aplicaciones de empresa son aquellas aplicaciones de software que facilitan diversas actividades dentro de una empresa.

Desde hace un tiempo que se conoce que la mejor solución en el desarrollo de sistemas es dividir las responsabilidades en diversas capas, lo que da como resultado una arquitectuta multicapa o distribuida. Así, en una aplicación de tres capas, por ejemplo, tendremos la capa de presentación responsable de atender las interacciones con el usuario, la capa de negocio responsable de las reglas de negocio (condiciones que se deben cumplir

para la ejecución de un proceso) y la capa de integración que provee acceso a los datos necesarios para la aplicación. Esta división permite que el contenido de cada capa sea cambiado independientemente.

Contenedores

El concepto de contenedor es primordial en la plataforma J2EE. Un contenedor provee de un entorno en el período de ejecución a los componentes de la aplicación (JSPs, servlets, o EJBs) que se ejecutan en él. Por ejemplo el contenedor EJB controla el ciclo de vida (creación y eliminación de componentes cuando sea necesario), la gestión de transacciones, seguridad, y la persistencia de los componentes EJB's que se ejecutan en él.

El hecho de que el componente se ejecute dentro de un contenedor es transparente para el cliente.

Configuraciones de despliegue

Una configuración de despliegue es una correspondencia entre la funcionalidad y los componentes de la aplicación y por lo tanto con los contenedores J2EE y los servicios que ofrece. En concreto, se trata de estructurar y distribuir la funcionalidad de la aplicación en capas, contenedores y componentes. Las más comunes son las que utilizan una capa, las que se centran en EJB's, las que se centran en la capa web y las multicapas.

Módulos J2EE

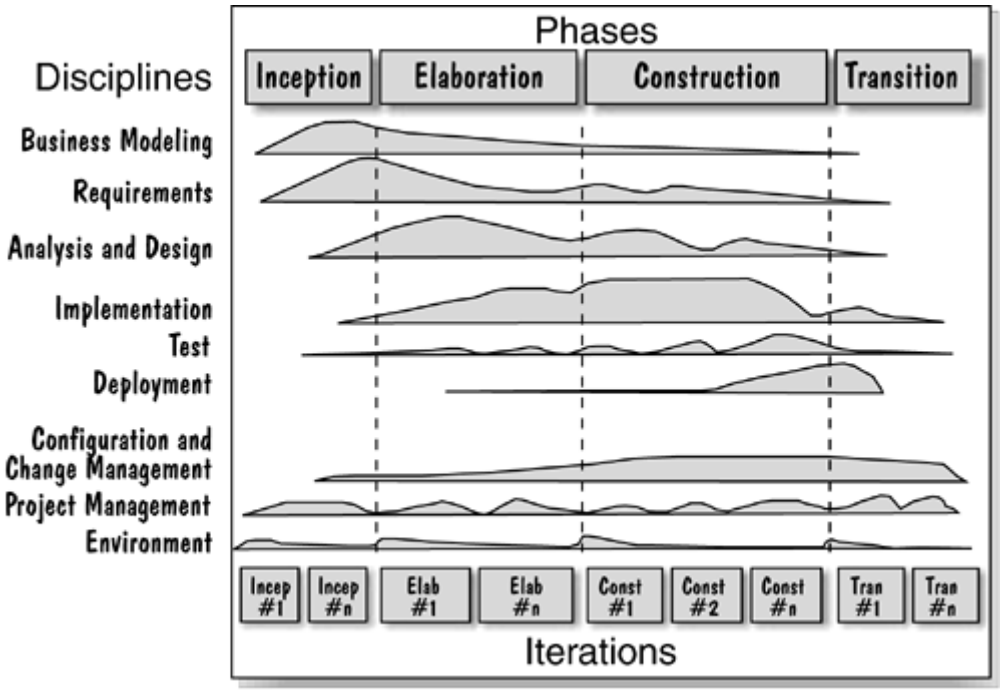
Todos los componentes de una aplicación son empaquetados en módulos J2EE, físicamente son archivos JAR. Un archivo JAR contiene uno o más ficheros de manera comprimida (del tipo ZIP). El uso de archivos JAR permite que los archivos relacionados se ejecuten como una unidad. Así, por ejemplo un módulo web es un paquete que contiene los componentes de la capa de presentación de la aplicación como son los archivos JSP's, Servlets y las clase Java necesarias.

Módulo J2EE	Contenido	Tipo de fichero	Descriptor de despliegue
Web module	JSPs, servlets, image files, static HTML files, Java classes	Web Archive (WAR)	WEB-INF\web.xml
EJB module	EJBs, Java classes	Java Archive (JAR)	META-INF\ejb-jar.xml
Resource adapter module	Resource adapters	Resource adapter Archive (RAR)	META-INF\ra.xml
Application client module	Java classes	Java Archive (JAR)	application-client.xml
J2EE application module	J2EE modules	Enterprise Archive (EAR)	META-INF\application.xml

RUP

El Rational Unified Process, RUP, es un framework o estructura para el proceso de desarrollo de software. Mediante una serie de pasos (disciplines) asigna tareas y responsabilidades en una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad, que satisfaga a los usuarios, dentro de un tiempo y presupuesto predecible, *"better software faster"*.

RUP se puede describir mediante el uso de dos dimensiones: tiempo y contenido. La figura siguiente muestra esta interpretación. El eje horizontal representa el tiempo y muestra los aspectos del ciclo de vida del proceso (se describe en términos de fases e iteraciones), el eje vertical representa el contenido y muestra las disciplinas, éstas se encargan de agrupar lógicamente el contenido del proceso.



Como enseña la figura anterior, la dedicación a cada disciplina cambia a lo largo del ciclo de vida del proyecto. Así, por ejemplo, en las primeras iteraciones se dedica más tiempo en los requerimientos y en las últimas la mayor parte del tiempo se corresponde con la fase de implementación.

Un proceso de desarrollo de software efectivo debe describir quién hace qué, cómo y cuando. RUP implementa exactamente esto en los términos de los siguientes conceptos clave:

- Roles (actores o responsables): Quién
- Artifacts (finalidad de una fase): Qué
- Activities (pasos en una fase): Cómo
- Fases, iteraciones, disciplinas y detalles de flujo: Cuando

UML

El *Unified Modeling Language* es un lenguaje gráfico para el modelaje y desarrollo de sistemas de software. Provee la infraestructura para la visualización y el modelaje de todas las fases en el desarrollo de software, desde los requerimientos a la especificación, la construcción y el despiegue.

La idea central al usar UML es capturar los detalles más significativos del sistema, de manera que el problema sea claramente entendido, y se identifique y construya una arquitectura y una implementación.

Con UML se pueden expresar complejas relaciones entre los bloques básicos de la aplicación. Estas relaciones pueden ser estáticas o dinámicas. Las

primeras tratan de aspectos estructurales del sistema, por ejemplo herencia, implementación de interfaces y dependencia entre clases. Las relaciones dinámicas tratan del comportamiento del sistema, esto es en tiempo de ejecución. Los mensajes intercambiados entre un grupo de clases para llevar a cabo cierta funcionalidad y el control de flujo dentro de un sistema, por ejemplo.

Ambos aspectos del sistema son capturados en diagramas UML. Hay muchos tipos de ellos y están organizados en áreas específicas del modelaje visual llamadas *Vistas* (views).

CAPÍTULO 3

REQUISITOS

Los requisitos son la especificación de lo que ha de hacer la aplicación, son descripciones del comportamiento, propiedades y restricciones del software que se va a desarrollar.

Juegan un doble papel:

a- Servir de base para un acuerdo entre los usuarios (empresa cliente) y los desarrolladores sobre el software que se va a crear. Esto significa que la documentación de los requisitos se debe realizar de una manera inteligible para dichos usuarios que no tienen porqué entender de programación.

b- Los requisitos son la información de partida para desarrollar el software, son la entrada a la etapa siguiente, el análisis.

Se han seguido los siguientes pasos para representar esta etapa:

Enunciado
Modelo de dominio
Casos de uso
Guiones
Glosario
Documentación textual de los casos de uso

Enunciado

Se trata de desarrollar una tienda virtual en una arquitectura distribuida con las siguientes características:

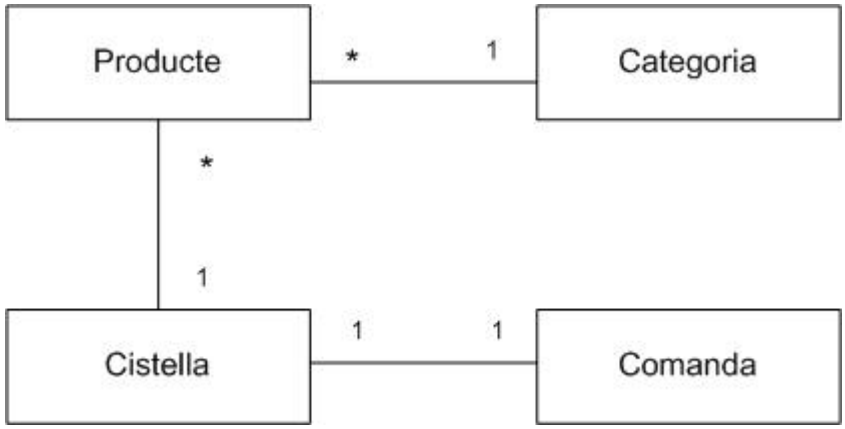
-Un catálogo de productos organizados por categorías. El cliente ha de poder navegar por las diferentes categorías y ver los productos que hay en cada una de ellas. Cada producto contendrá cierta información.

-Un buscador de productos que permita buscar productos directamente. A partir de la búsqueda se debe mostrar un listado paginado que cumpla con los criterios.

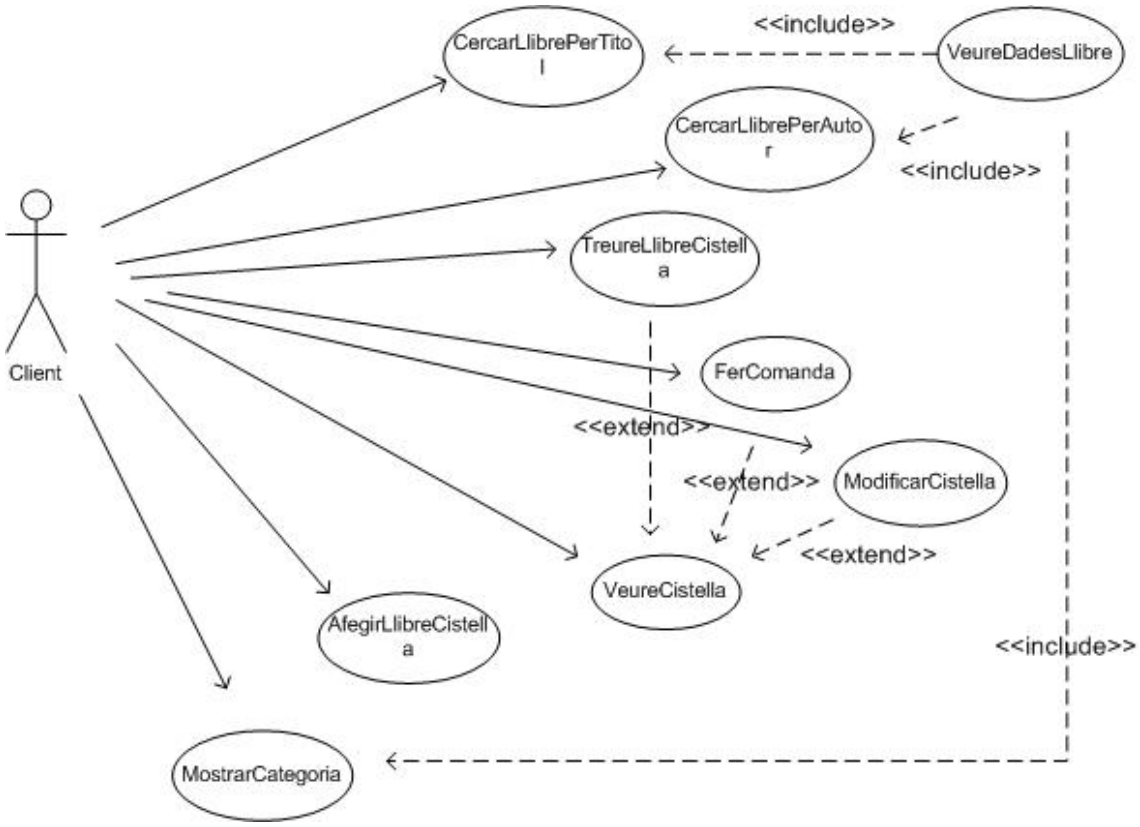
-Cesta de la compra. El usuario ha de poder seleccionar un producto y añadirlo a la cesta. La cesta debe tener las funciones típicas (agregar elemento, borrar..)

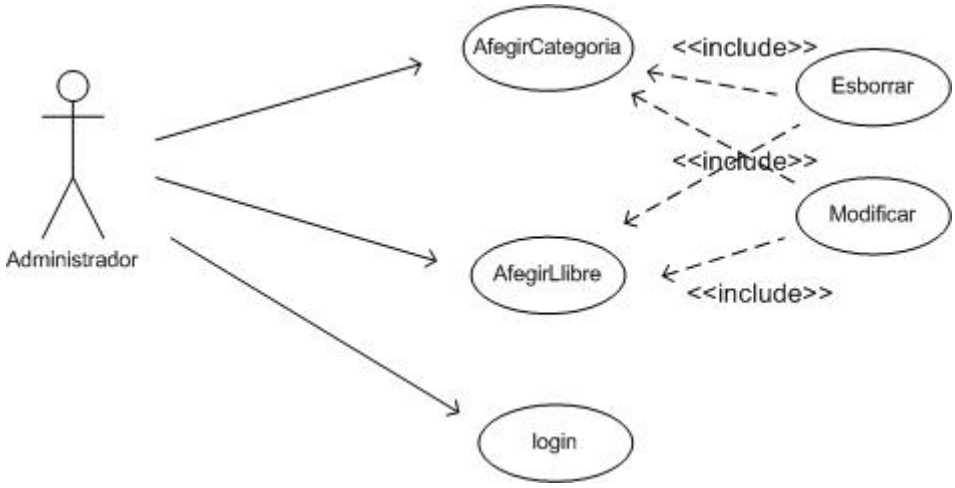
-También se debe desarrollar un gestor de contenidos para la tienda. Será un módulo que permita administrar los productos que se venden en la tienda, es decir, agregar, borrar y modificar la información de cada uno. Lo mismo respecto a las categorías. Se accederá a él mediante un login previo cuyo rol debe ser “administrador”.

Modelo de dominio



Casos de uso





Guiones

Guión del cliente:
Entra en la web donde se le presentan las categorías disponibles de los diferentes productos, un cuadro de búsqueda con diferentes criterios y una selección de productos bajo el título de 'novedades'. Los resultados de las

búsquedas y las listas de los productos en una categoría se le presentarán paginados.
A partir de aquí podrá agregar, borrar y modificar los productos que escoja de su cesta particular, la cual se activa la primera vez que agregue un producto. Finalmente puede hacer el pedido.

Guión del administrador:
Desde la aplicación que le corresponde podrà agregar, borrar y modificar categorías y productos. La eliminación de una categoría implica la eliminación de todos los productos que contenga. A esta aplicación sólo podrá acceder un usuario con el rol de administrador.

Glosario

Tienda virtual, aplicación web que permite seleccionar productos de un catálogo categorizado y agregarlos a una cesta de la compra, para posteriormente realizar un pedido que se entregará en el domicilio del cliente.

Cesta, conjunto de productos escogidos por un usuario con la intención de comprar.

Pedido, acción de comprar los productos de la cesta.

Documentación textual de los casos de uso 'Ver Categoría' y 'Ver Cesta':

Caso de uso: Ver Categoría

Resumen de la funcionalidad: Permite ver los productos contenidos en una categoría.
Papel dentro del trabajo del usuario: habitual.
Actores: Cliente
Casos de uso relacionados: Ninguno
Precondición: Ninguna
Postcondición: Se muestra una lista paginada con los contenidos de la categoría seleccionada.
Proceso normal principal:

1. El usuario clica sobre el enlace que nombra la categoría.
2. El sistema devuelve la lista con los productos que contiene dicha categoría.

Alternativas de proceso y excepciones:
Ninguna

Caso de uso: Ver cesta

Resumen de la funcionalidad: Permite ver los productos que han sido agregados a la cesta de la compra
Papel dentro del trabajo del usuario: habitual
Actores: Cliente
Casos de uso relacionados: *Pedido*, *Borrar Producto de la cesta*, *Vaciar Cesta*.
Precondición: ninguna
Postcondición: ninguna (es una consulta)
Proceso normal principal:

1. El sistema muestra al usuario el listado de productos de la cesta de la compra. Para cada producto se muestra el nombre, la cantidad de unitades del producte seleccionado, el precio unitario y el precio total (precio unitario * número de unidades). También se muestra el precio total de la cesta de la compra.

Alternativas de proceso y excepciones:
2a. El usuario selecciona un producto y la opción eliminar
El sistema ejecuta el caso de uso borrar producto
2b. El usuario selecciona 'pedido'
El sistema ejecuta el caso de uso Pedido
2c. El usuario selecciona borrar contenidos.
El sistema ejecuta el caso de uso Vaciar cesta

Así pues, el producto obtenido de esta fase es una documentación semi-formal acerca del software que se implementará en forma de casos de uso por un lado y de tareas de los usuarios, por el otro, y que servirá como punto de partida para la etapa posterior. En concreto, y fundamentalmente se han encontrado los tipos de usuarios que tendrá el sistema y qué procesos podrán llevar a cabo.

Las mejoras que se puedan realizar al respecto implican ulteriores entravistas con el cliente y funcionalidades extra no requeridas.

CAPÍTULO 4

ANÁLISIS

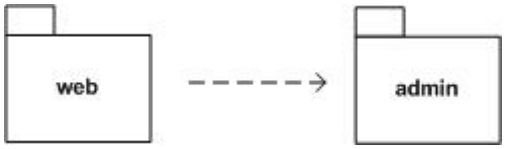
El primer objetivo del análisis es traducir los requisitos a un lenguaje más formal, que, en el método seguido, son los modelos y diagramas de UML, con lo cual se obtendrá el *modelo de análisis.*
Un segundo objetivo es la identificación de una clases fundamentales que serán la base de la implementación del software. Y un tercer objetivo será la expresión de los casos de uso en términos de estas clases.

Se han seguido los siguientes pasos para representar esta etapa:

Paquetes de análisis
Especificación de las clases de análisis
Diagramas de clases
Diagramas de colaboración simplificados
Diagramas de secuencia
Análisis de la interficie de usuario

Paquetes de análisis

Se han hecho dos paquetes teniendo en cuenta los dos actores del sistema:



Especificación de las clases de análisis

Del enunciado y de los casos de uso se pueden ver las clases siguientes:

Llibre: idLlibre, idCategoria , titol, autor, descripció, preu

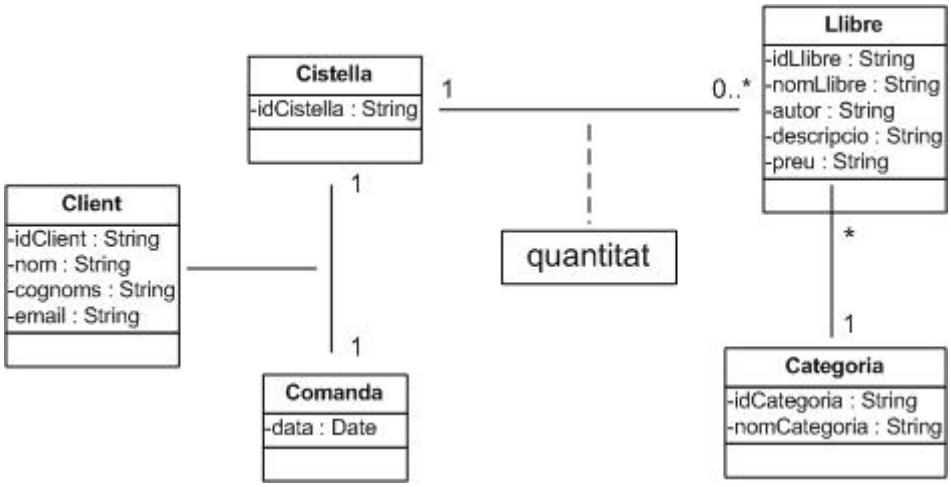
Cistella: idCistella, idLlibre

Comanda: idCistella, idClient, data

Categoria: idCategoria, nomCategoria

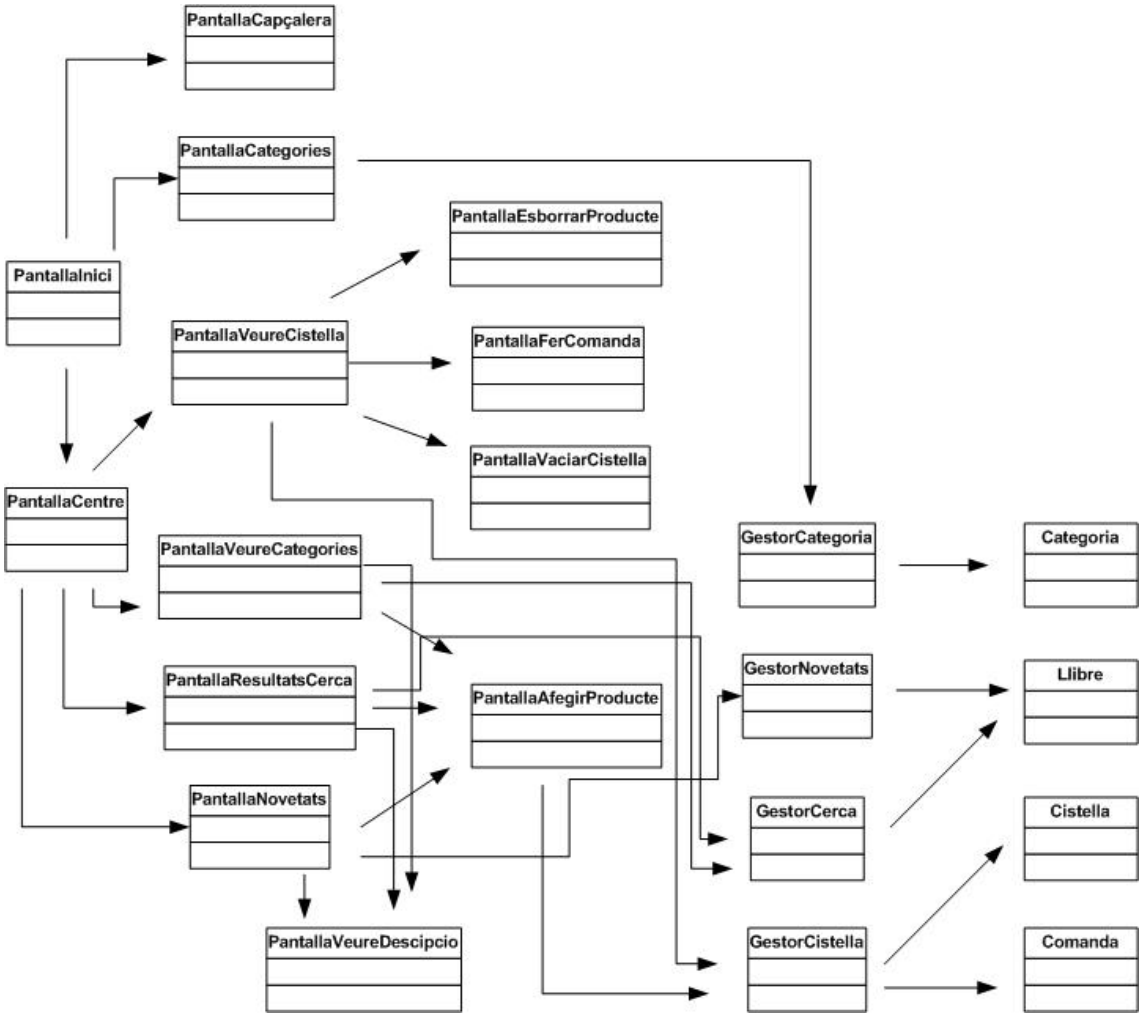
Client: idClient, nomClient, cognoms, email

No se han encontrado relaciones de herencia ni agregación, sí de asociación:

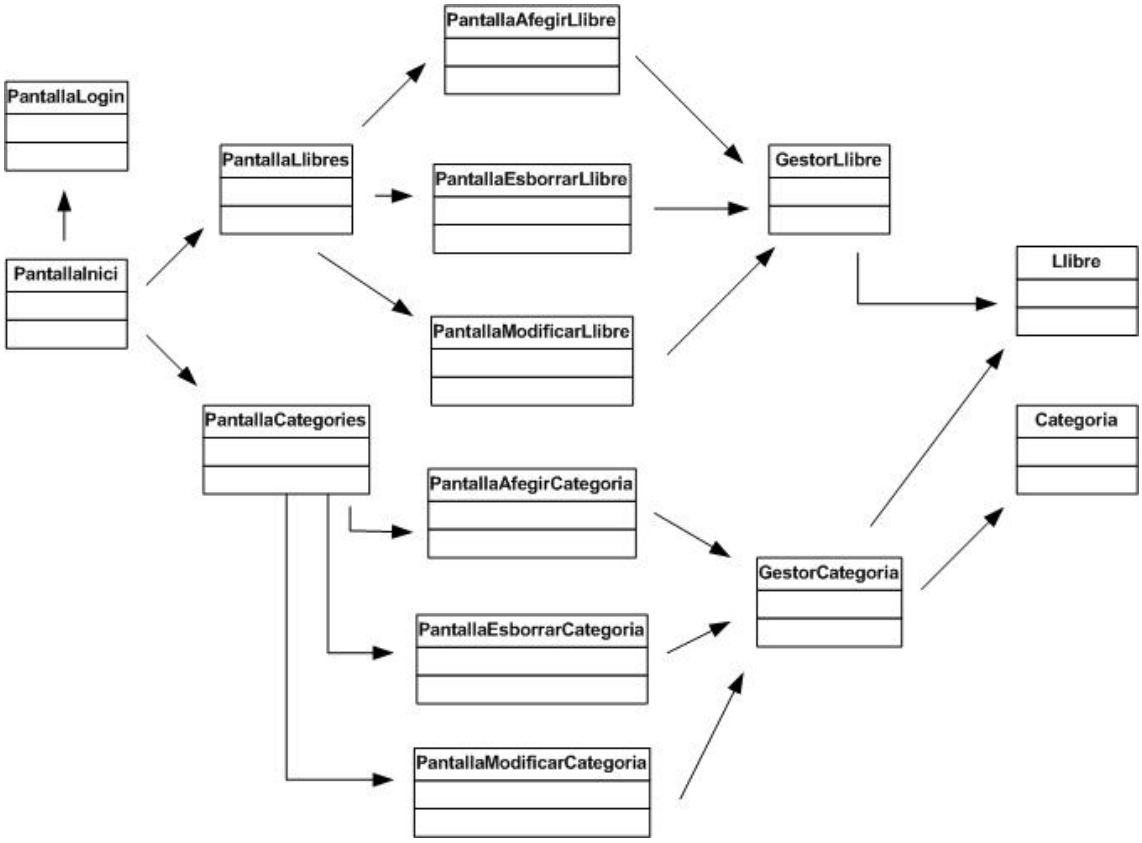


Diagramas de clases

para el web:



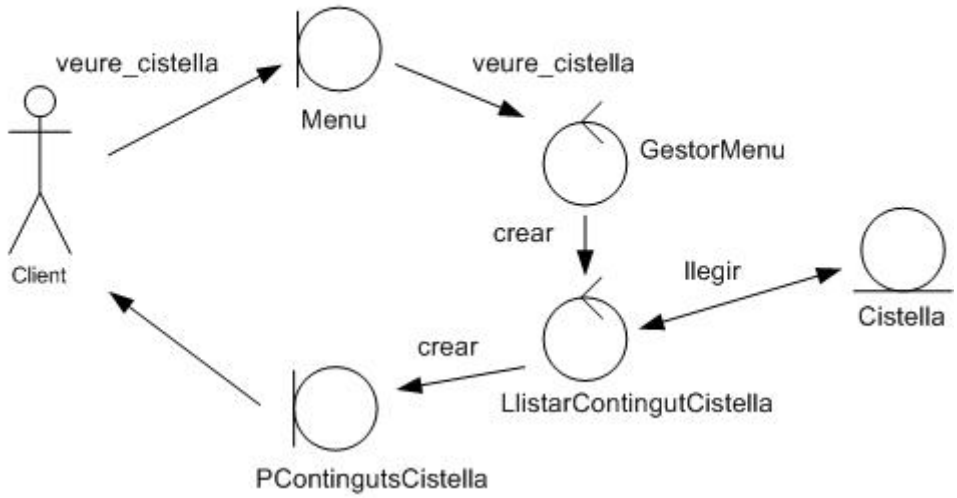
para el gestor:



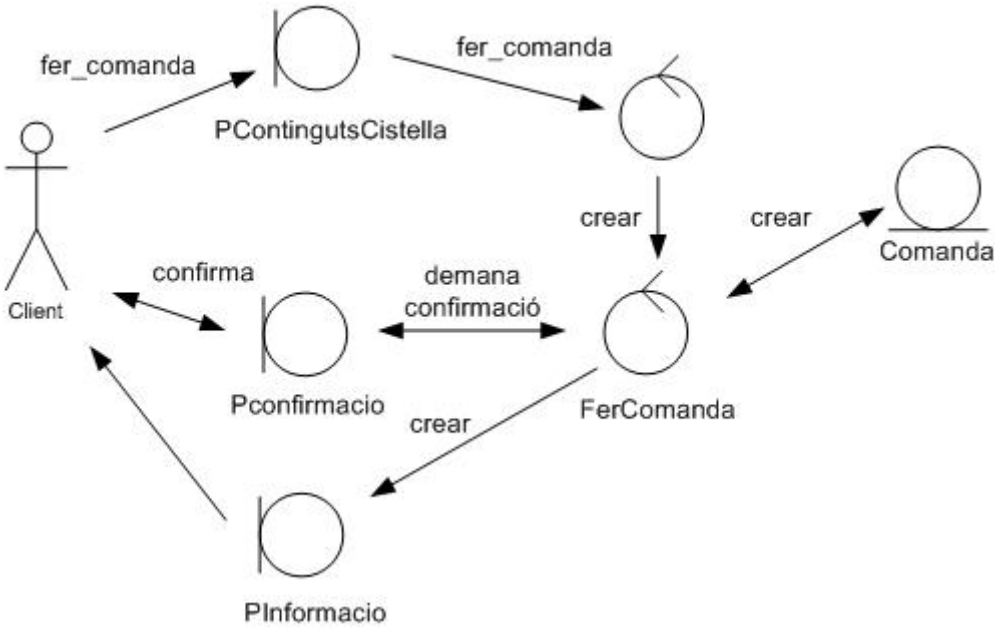
Diagramas de colaboración simplificados.

Se incluye el diagrama de dos operaciones para cada rol, el resto sigue la misma lógica y son totalmente similares:

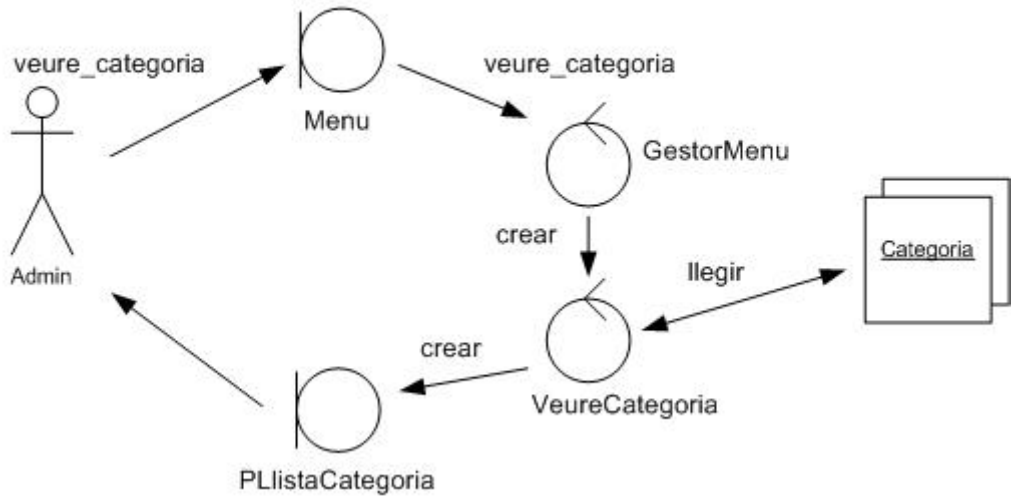
VeureCistella

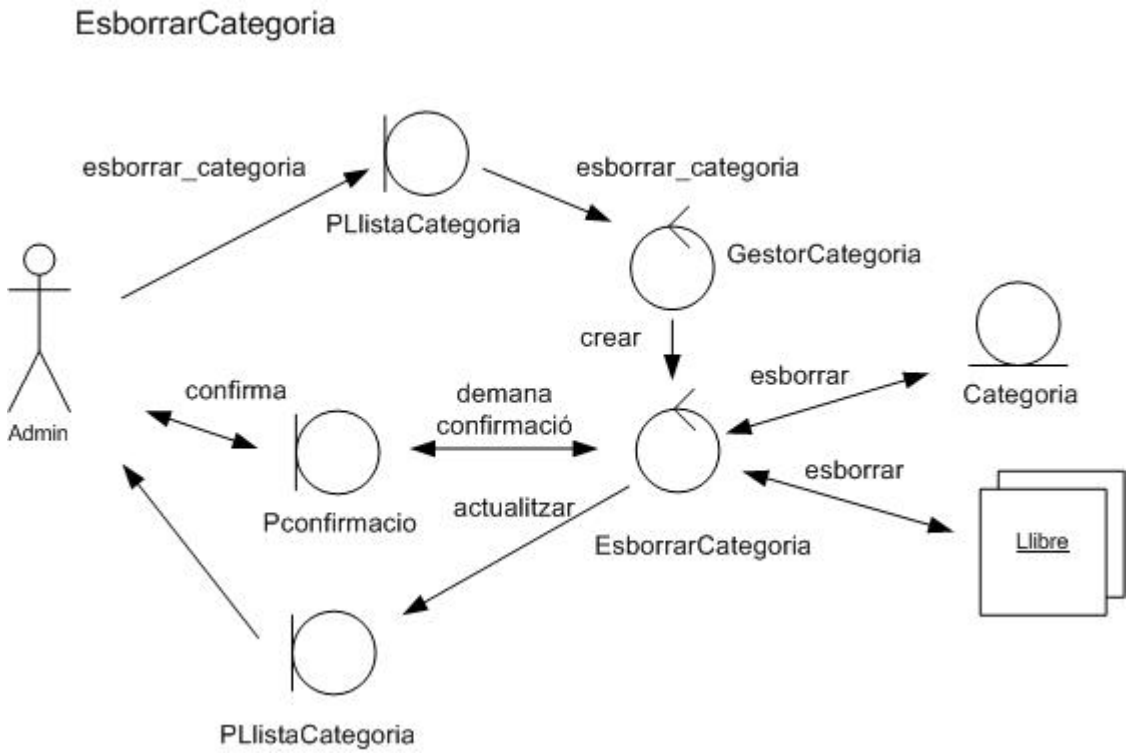


FerComanda



VeureCategories

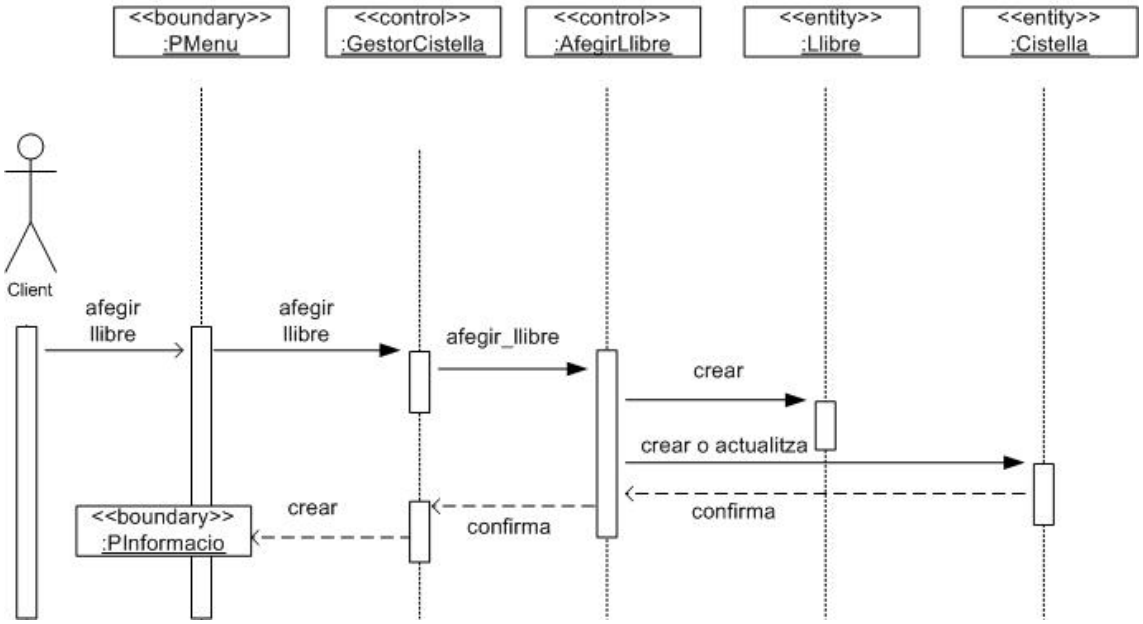




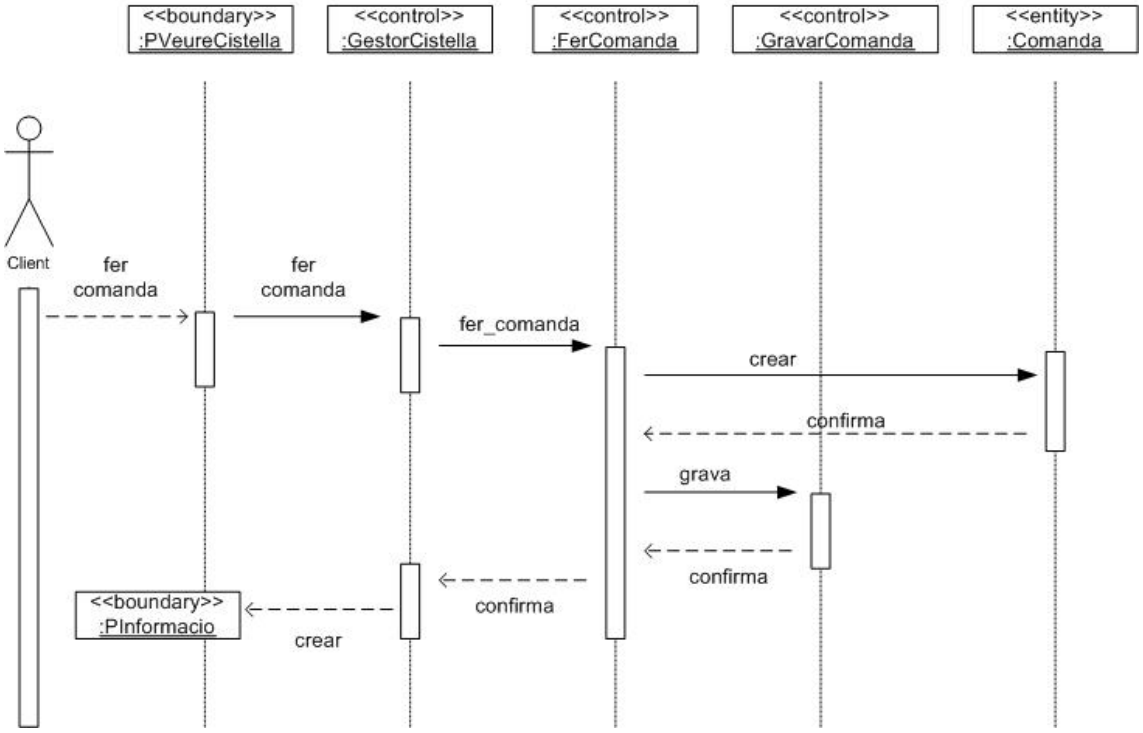
Diagramas de secuencia

Vemos la interacción de las diferentes clases en dos diagramas de secuencia, se podrían dibujar tantos como sean necesarios para entender el funcionamiento del sistema:

AfegirLlibre

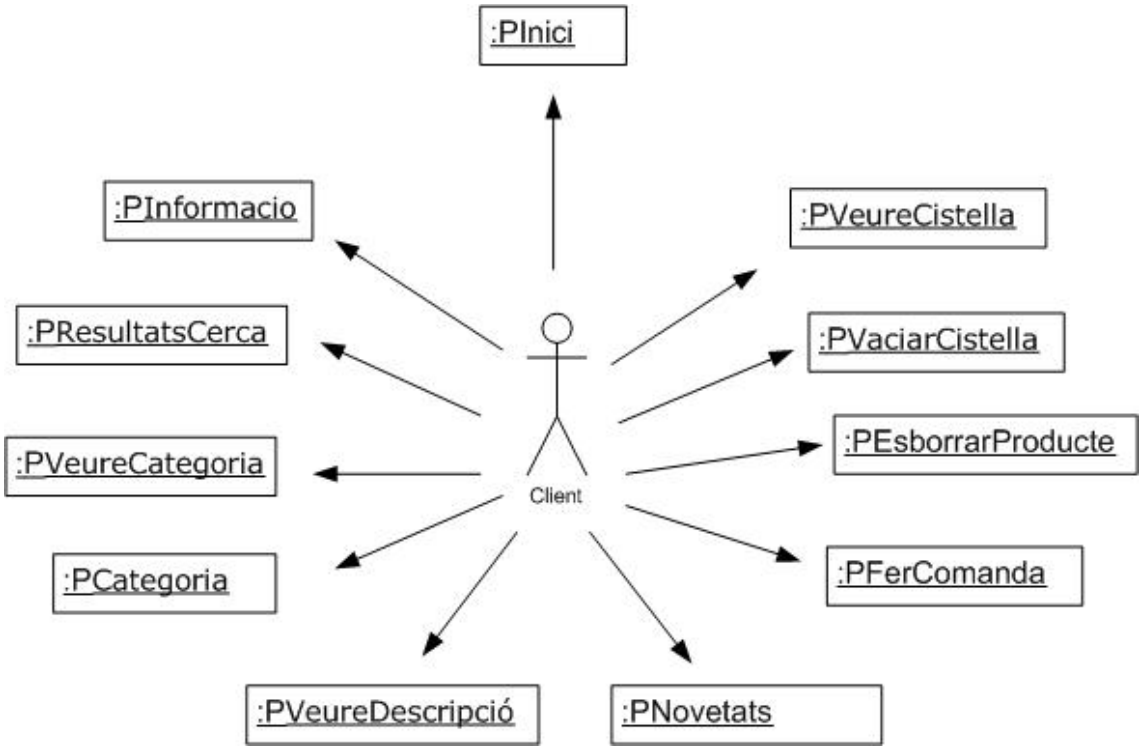


FerComanda

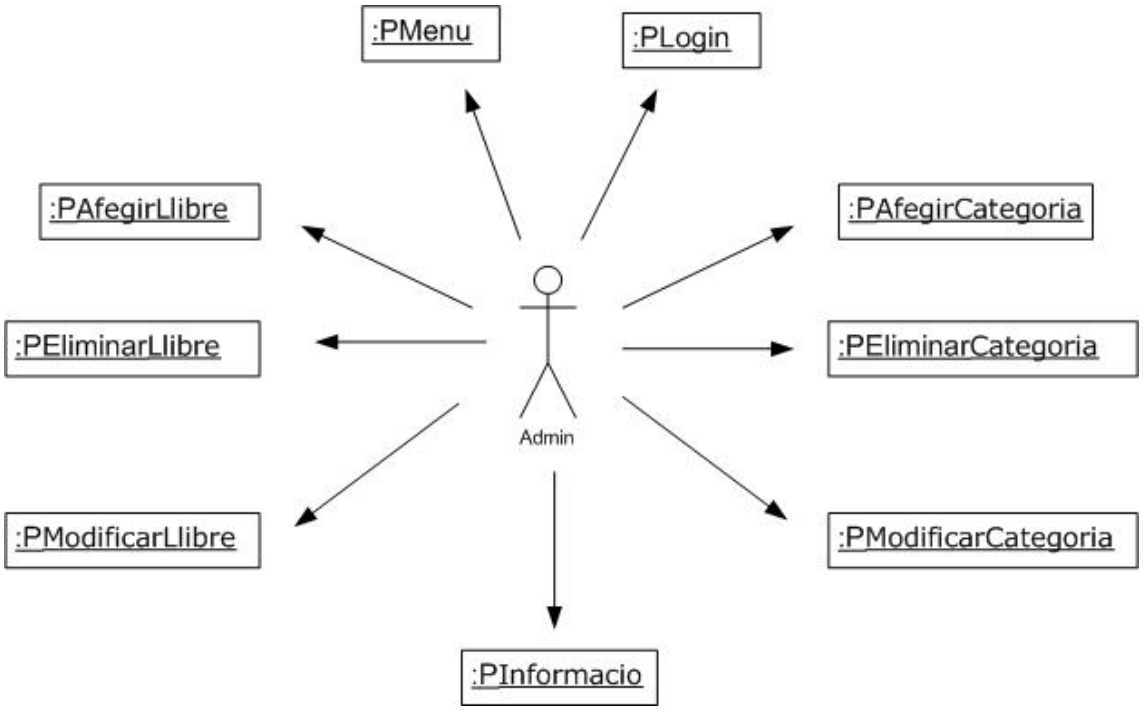


Analisis de la interficie de usuario

Pantallas que ve el cliente:



Pantallas que ve el administrador:



En este momento, ya se dispone del principio (pantallas) y el fin de la aplicación (clases, algunas de ellas serán trasladadas a la base de datos).

También se dispone del flujo de las relaciones, en el próximo capítulo se decidirá cómo se implementarán en el lenguaje escogido para la aplicación, esto determinará la *arquitectura del sistema*.

También, la interfaz tomará forma a manera de bocetos, los cuales indicarán la disposición de los casos de uso en la pantalla.

CAPÍTULO 5

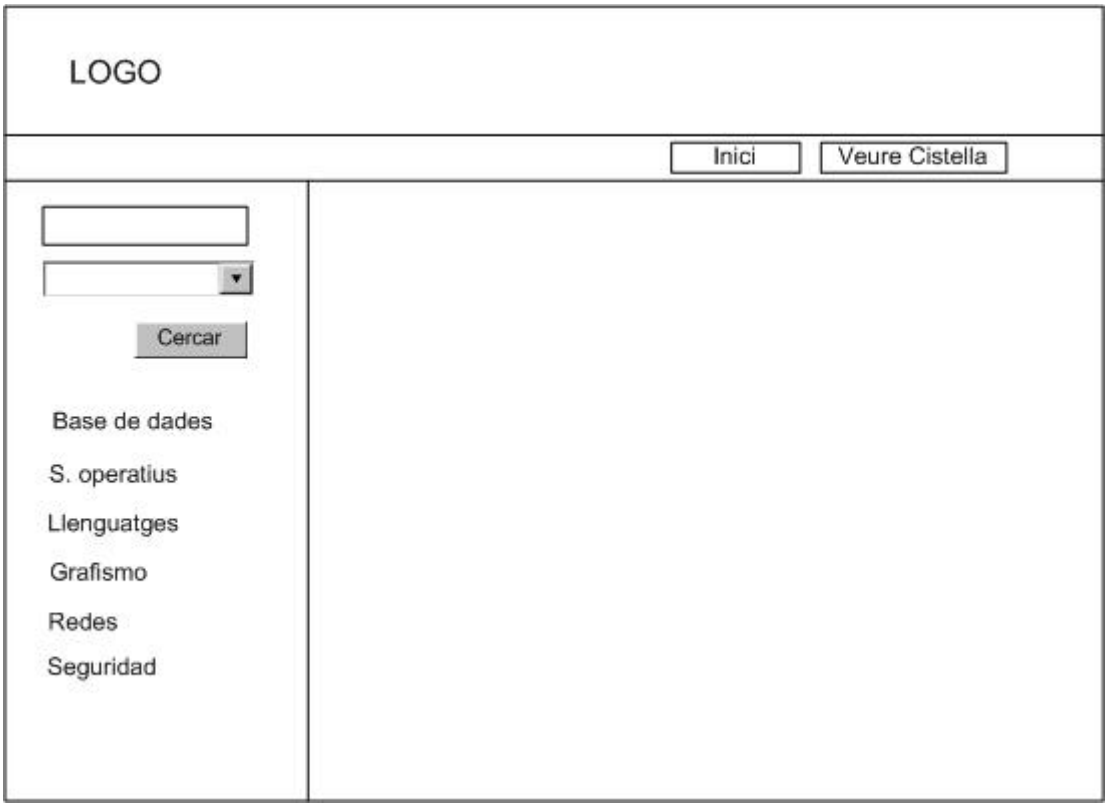
DISEÑO

La finalidad del diseño es la de servir de puente entre el análisis y la implementación. En esta etapa ya se presentan resultados concretos de la apariencia interna y externa del software traducidas al lenguaje y las herramientas de desarrollo.

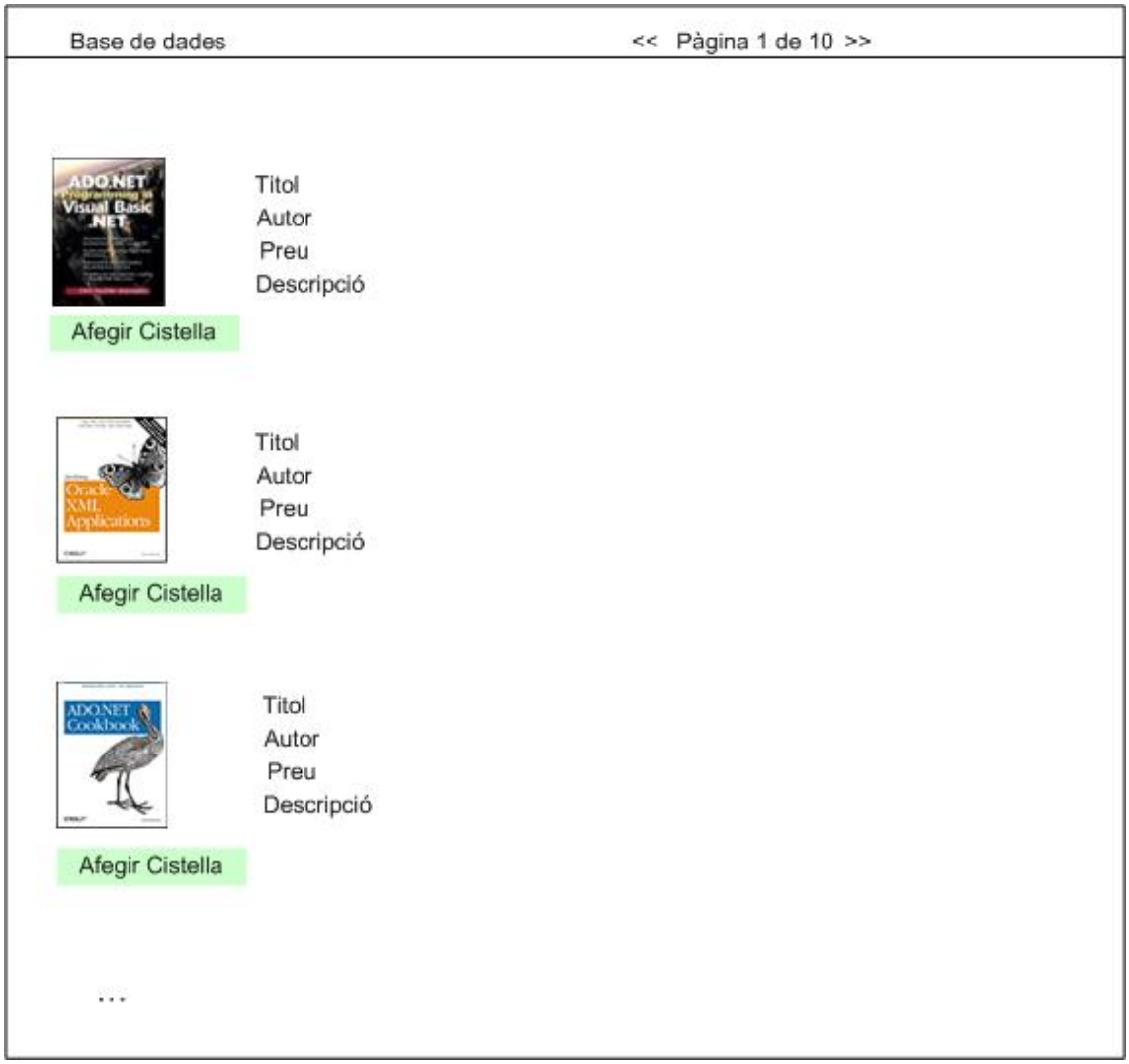
Se han seguido los siguientes pasos para representar esta etapa:

Diseño Frontera

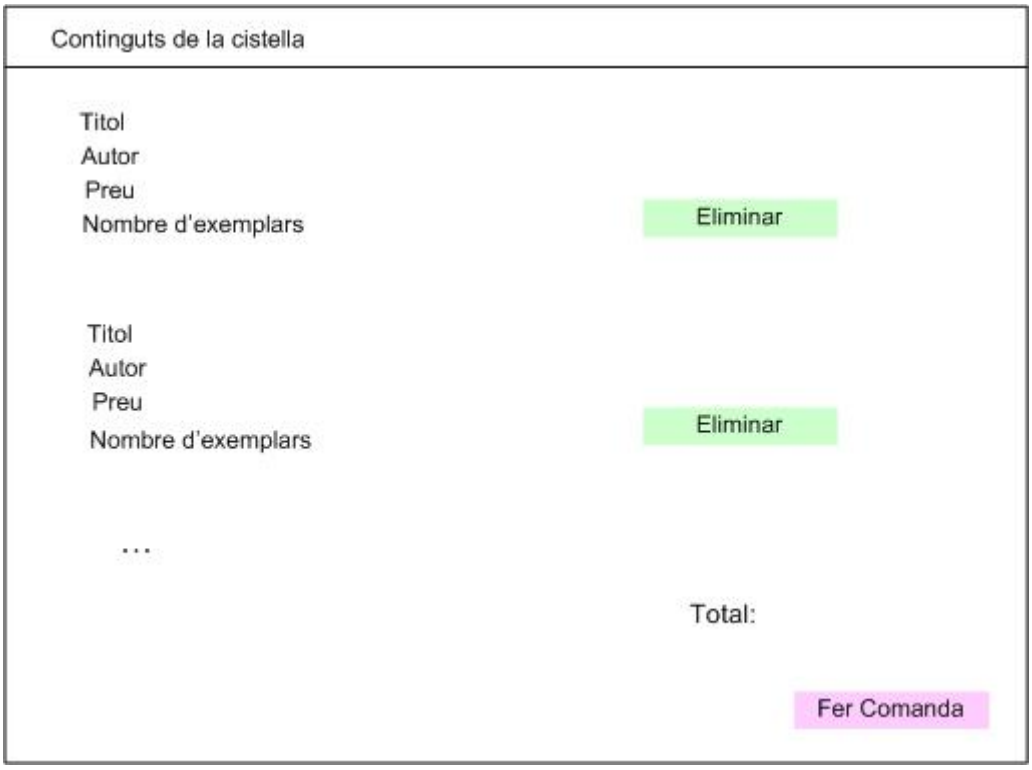
Pantalla de entrada web formada por 3 frames: cabecera, lateral y central.



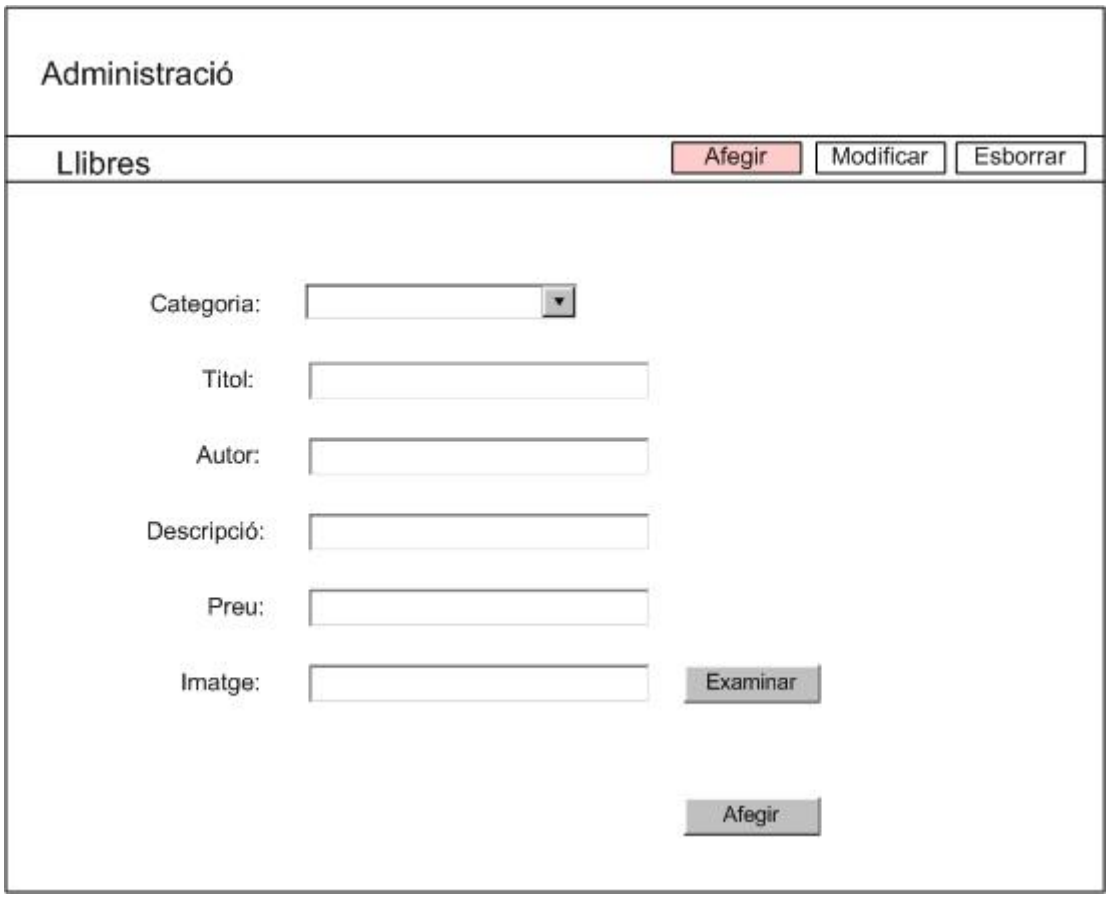
Al clicar sobre Base de dades, por ejemplo, en el frame central aparece:



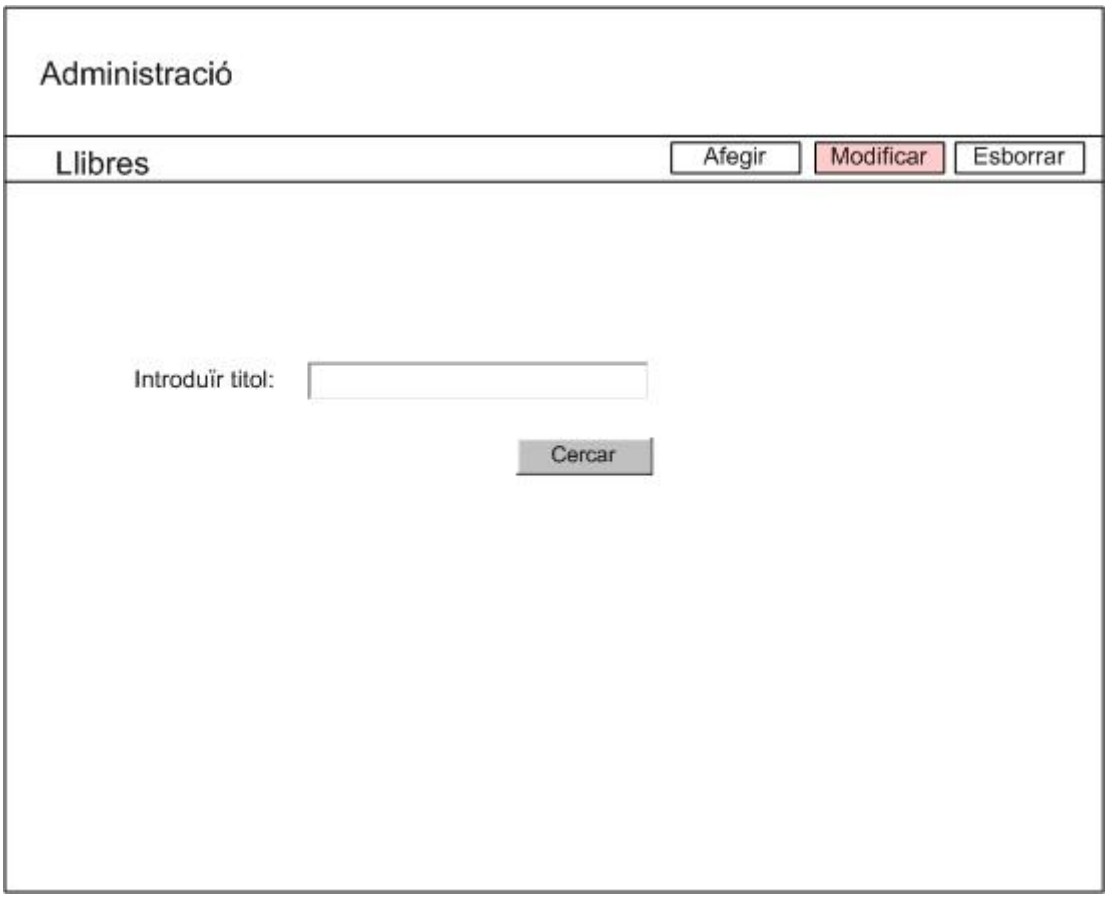
Al clicar en cesta se muestran los contenidos:



El cliente de la aplicación de administración serà web, la pantalla de inicio le permitirá escoger entre Categoria i Llibres. Por ejemplo, en la pantalla Llibres, por defecto aparece la opción agregar:



Para modificar los datos de un libro (o para eliminar uno) hay que introducir el título, los resultados de la búsqueda mostrarán una lista de títulos por orden descendiente de similitud.

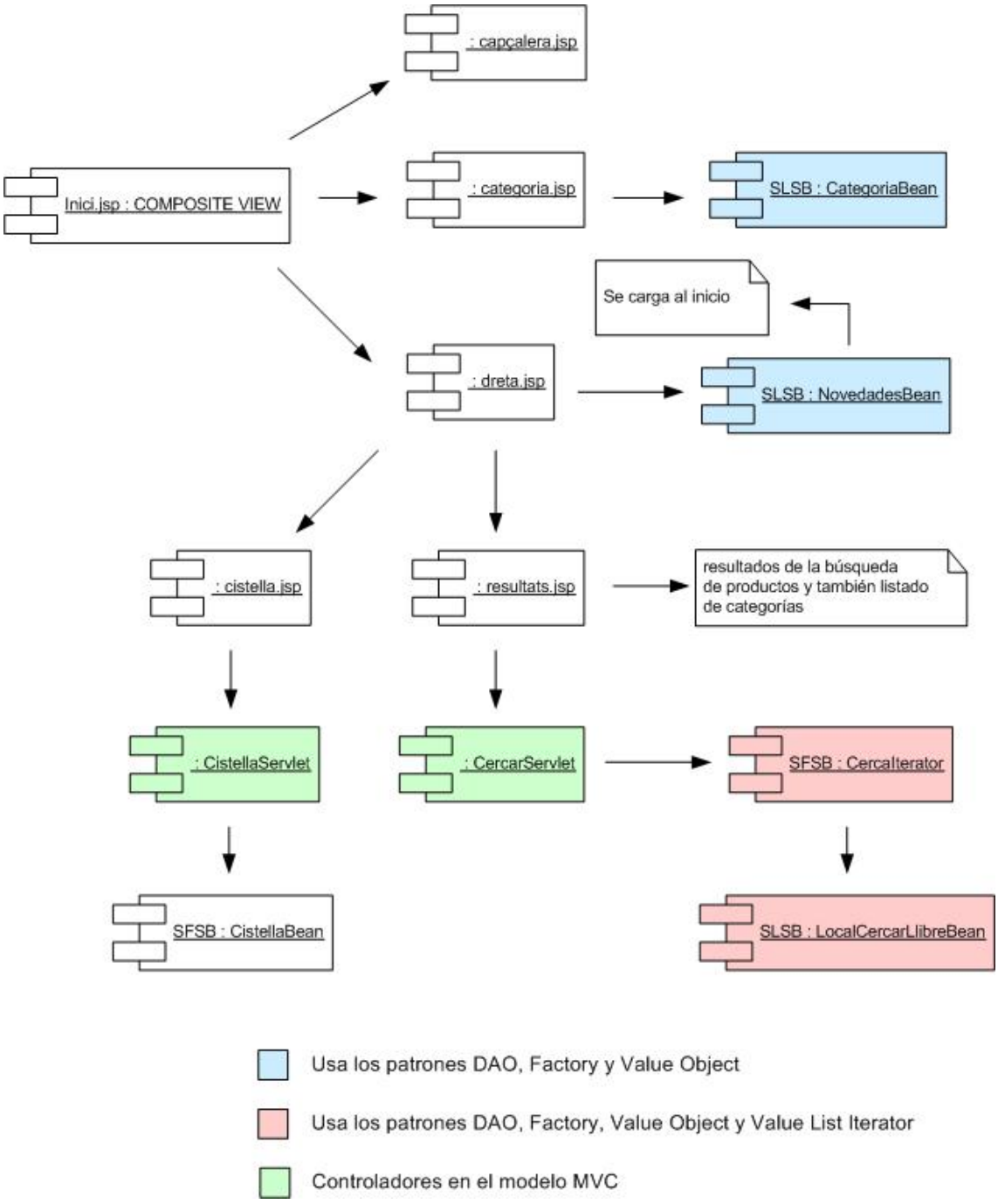


Diseño Control
Web

Se usará el modelo de diseño de aplicaciones 2, Model-View-Controller.
Se programarán dos servlets, uno encargado de las búsquedas y el otro de los contenidos de la cesta.
El frame izquierdo de la pantalla de inicio se cargará directamente por medio de una consulta de la pàgina jsp al bean correspondiente, de la misma forma que la página complementaria 'novedades' que aparece inicialmente en el frame central.

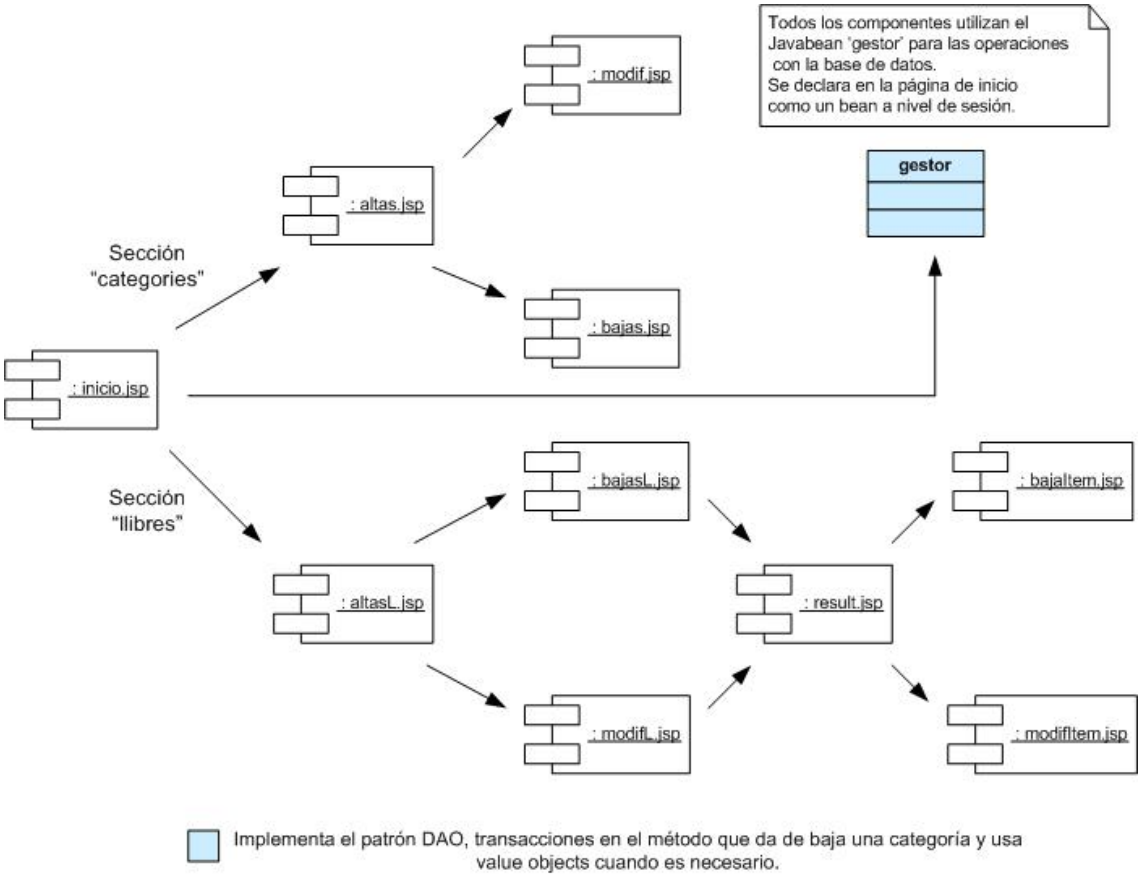
Estos servlets se comunicarán con los EJBs encargados de la lógica de negocio, actualizarán variables en el ámbito correspondiente y redireccionarán hacia una página JSP, la cual leerá las variables anteriores.

Diagramas de componentes del web:



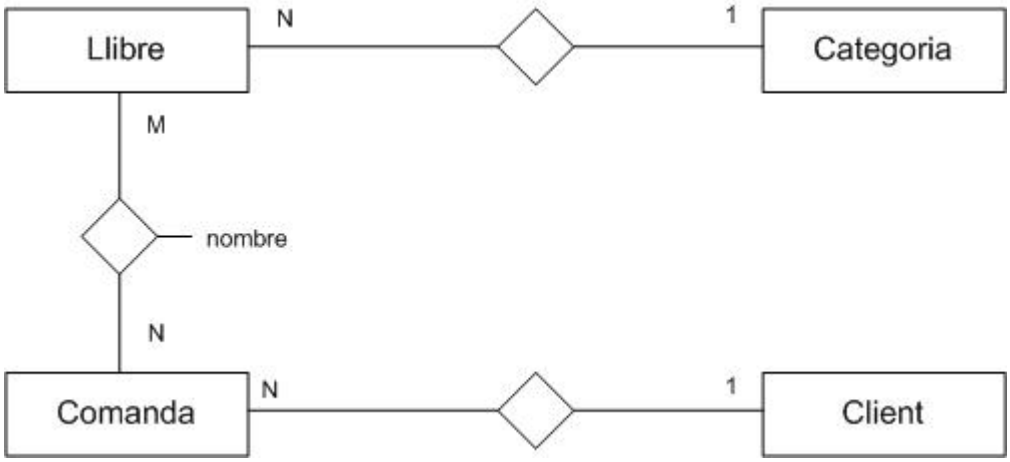
Diagramas de componentes del gestor:

Teniendo en cuenta que la administración la lleva una única se usará el modelo de diseño de aplicaciones 1 (1.5 en ciertos libros). Éste carga la lógica de negocio en Javabeans.



Diseño Entidad

Base de datos



index

Las tablas son las siguientes:

Taula TCategoria

idCategoria: Integer
nomCategoria: String
nomCategoria: unic
Clau primària: idCategoria

Taula TLlibre
idLlibre: Integer
idCategoria: Integer
titol: String
autor: String
preu: Integer
descripcio: String
imatge: String
Clau primària: idLlibre
Clau forana: nomCategoria

Taula TClient
IdClient: Integer
nom: String
cognoms: String
email: String
Clau primària: IdClient

Taula TComanda
idComanda: Integer
idClient: Integer
data: Date
Clau primària: idComanda
Clau forana: idClient

Taula TVista
idComanda: Integer
idLlibre: Integer
nombre: Integer
Clau primària: idComanda, idLlibre

Identificación de patrones

La página de inicio web está formada por 3 frames, se puede decir que se está usando el patrón de diseño COMPOSITE VIEW.

En la capa de presentación, respecto a la interacción con el cliente, es usará el patrón de diseño MVC con el objectivo de separar al máximo la presentación de la lògica de negocio.

En la capa de negocio se usará el patrón de diseño VALUE LIST ITERATOR para la paginación; SESSION FAÇADE, para acceder al bean de entidad desde uno de sesión; VALUE OBJECT, para minimizar la comunicación con el servidor.

En la capa de datos se usará el patrón de diseño DAO (el cual usa el patrón FACTORY para su implementación) para encapsular código, en principio, no portable, como es el de acceso a diferentes SGBD.

Diagrama del patrón Value List Iterator:

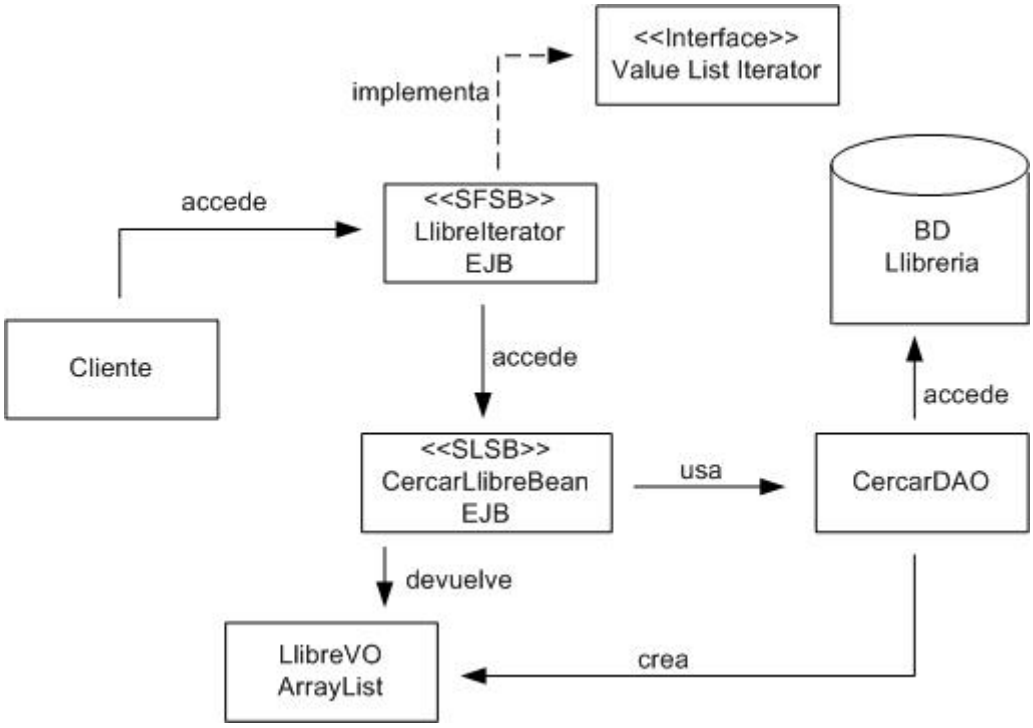
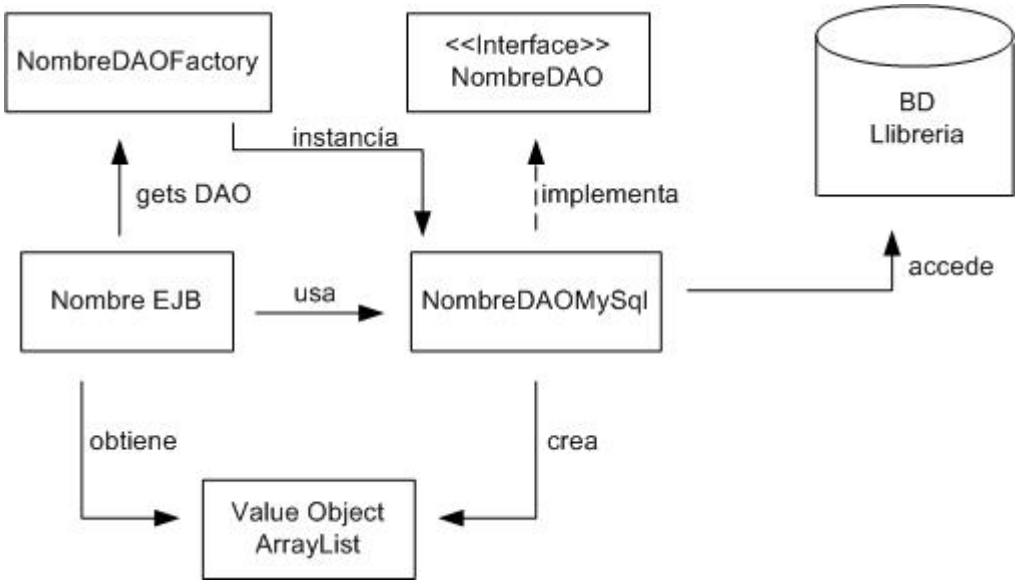


Diagrama del patrón DAO:



CAPÍTULO 6

IMPLEMENTACIÓN

Conceptos y claves de la implementación

Beans

Un bean de sesión sin estado permite que múltiples clientes compartan una sóla instancia. Los clientes pueden realizar múltiples peticiones a bean de sesión sin estado, pero éste no guarda ningún tipo de dato referente al cliente.

Un bean de sesión (con o sin estado) generalmente implementa un proceso de negocio. Un proceso de negocio es una serie de tareas que conforman el conjunto de reglas del negocio de la empresa (restricciones y condicionantes). El bean de sesión debe seguir estas reglas y asegurarse de que el proceso cumple las condiciones.

En estos procesos se manipulan datos, pero los resultados de la manipulación son volátiles, simplemente los resultados se envían al cliente.

La principal ventaja de un bean de sesión sin estado es que el contenedor EJB puede hacer que una sóla instancia del bean esté disponible para múltiples clientes. Por otro lado, un bean de sesión con estado, debe mantener el estado de cada cliente. Por lo tanto el contenedor EJB no puede permitir el acceso de múltiples clientes a la misma instancia y tendrá que pasar el bean a estado pasivo cuando tenga que liberar recursos enviándolo a la memoria secundaria. Esto significa que un bean de sesión sin estado

puede ser más eficiente que uno con estado dado que impone menos restricciones al contenedor.

Un bean sesión con estado guarda datos específicos del cliente durante una sesión. Estos datos se guardan en variables de instancia del bean. El bean mantiene el estado conversacional entre el cliente y el bean. El contenedor asigna una instancia del bean a un único cliente. Cuando la sesión se termina, se libera el estado del bean y la instancia deja de existir.

Para crear un bean de sesión de empresa se necesita crear al menos una clase y dos interfaces: una interface que extienda la interface EJBHome (home interface), una interface que extienda la interface EJBObject (remote interface), y una clase que implemente la interface SessionBean (bean implementation class). Cada una de estas partes juega un rol diferente en la arquitectura EJB y tiene ciertas restricciones que deben ser seguidas por el desarrollador.

En un bean de sesión el rol de la interfaz home es proveer un método create() mediante el cual el cliente crea el bean de empresa. El cliente usa el *lookup and naming service* (JNDI) que proporciona el servidor de aplicaciones para encontrar una referencia (“handle”) a la interface home. Cuando el cliente llama al método create() del objeto que implementa la interfaz el contenedor EJB instacía el bean.

Los procesos de negocio realizados por un bean de empresa se llaman métodos de negocio. Los clientes acceden al método create() a través de la interfaz home pero a los métodos de negocio se llega mediante la interfaz remota.

Todos los métodos de negocio en la interfaz remota deben especificar ‘RemoteException’ en la cláusula throws.

La clase de implementación de un bean de sesión debe implementar la interfaz SessionBean. También debe proporcionar los métodos ejbCreate(), ejbRemove(), ejbActivate(), ejbPassivate(), and setSessionContext(). El método ejbCreate() es la implementación del método create() de la interfaz home.

Pasivación y activación en beans sesión con estado

En el entorno J2EE, las aplicaciones que se ejecutan utilizan muchos recursos del sistema. El contenedor EJB, el cual controla el ciclo de vida y el estado activo del bean, puede necesitar recuperar los recursos adquiridos por el bean sesión con estado. Estos recursos incluyen memoria del sistema, conexiones a sockets y conexiones a bases de datos. En este momento el contenedor serializa el estado del bean y lo guarda en memoria secundaria. Cuando necesita activarlo, lee su estado de esta memoria secundaria y lo deserializa, este proceso se llama activación. El contenedor EJB realiza esta labor de forma silenciosa y transparente.

A veces el contenedor EJB no puede serializar el objeto debido, por ejemplo, a que sus variables de instancia no implementan la interfaz Serializable y por lo tanto el contenedor no puede guardar y recuperar su estado. En este caso debe ser el desarrollador quien proporcione el código para reinicializar el bean correctamente.
La interface del bean de sesión proporciona los métodos ejbPassivate() y ejbActivate(), los cuales permiten al desarrollador controlar la pasivación y activación del bean de sesión.
El contenedor invoca ejbPassivate() justo antes de pasivar la instancia y a ejbActivate() justo después que la reactiva. El desarrollador pone el código de estos métodos en la clase de implementación del bean.

En general, se debe diseñar el método ejbPassivate() para deshacer o librar las acciones o recursos realizadas o adquiridas por el método ejbActivate(). Por ejemplo, cerrar una conexión a la base de datos en ejbPassivate() si se ha adquirido en el método ejbActivate().

El contenedor EJB nunca necesita pasar al estado de pasivación un bean de sesión sin estado, dado que una vez retorna al cliente la invocación, el contenedor puede asignar la instancia a otro cliente. Este forma de compartir permite que recursos caros sean distribuidos entre varios clientes. Si el contenedor necesita recuperar recursos, simplemente destruye la instancia.

El trabajo real y poder de la informática de empresa reside en la capacidad de crear, borrar y actualizar registros en la base de datos con la confianza de que los datos sean siempre consistentes y sincronizados con todos los clientes y también con el SGBD. Para esta tarea se usan beans de entidad. En la arquitectura J2EE, se definen los datos de negocio en beans de entidad. Estos beans trabajan bajo la gestión del contenedor EJB que proporciona la integridad de los datos.

Un bean de entidad es una representación en memoria de datos persistentes. El mecanismo de persistencia es, o bien, codificado por los desarrolladores (bean-managed persistence) o bien proporcionado por el contenedor (container-managed persistence). Para ambos el contenedor gestiona su ciclo de vida y los mantiene sincronizados con la base de datos. Dado que múltiples clientes pueden compartir los beans de entidad (y posiblemente alterar su estado), el contenedor y el desarrollador deben asegurarse que cualquier actualización se haga dentro de la transacción apropiada. Los beans de entidad deben contener un mínimo de lógica, su ocupación son los datos.

Al igual que los beans de sesión, los de entidad pueden tener una interfaz remota o local (o ambas). Para los de persistencia gestionada por el bean es conveniente implementar la interfaz remota para el testing y la local para producción. Es recomendable acceder a ellos mediante un bean de sesión, de manera que las llamadas sean locales, mejorando así el rendimiento. Esto evita la sobrecarga de RMI y alivia el tráfico de la red. Al front end del bean de sesión se le ha dado el nombre de patrón Session Façade.

Los beans de entidad tienen una clave primaria única que se corresponde con la clave primaria de la base de datos.

Lo beans de entidad gestionados por contenedor requieren interfaces local y localHome.

Excepciones

Excepciones a nivel de aplicación son usualmente errores no fatales producidos por inconsistencias en la entrada de datos o por otra situación recuperable.
Por otro lado, las excepciones a nivel de sistema son generalmente una condición no recuperable, tales como el fallo al obtener una conexión a la base de datos, la imposibilidad de cargar una clase, o el fallo de una consulta a la base de datos.

Patrones

El patrón value object es una clase que encapsula una serie de datos útiles. Esto evita el realizar múltiples llamadas remotas en un EJB para guardar o recibir información. No implementan ningún método, sólo setters y getters para acceder a los datos encapsulados. Por lo tanto son convenientes y eficientes.

El patrón Data Acces Object (DAO) introduce un más profundo nivel de abstracción en cualquier EJB que acceda a base de datos. Primero se crean los métodos necesarios para acceder a la base de datos. Segundo, se define una clase Factory con un método estático que instancía una implementación de la interfaz abstracta. Esta clase Factory realiza una búsqueda (JNDI) para obtener el nombre de la clase de implementación. Este nombre se encuentra en el deployment descriptor del EJB y es posible cambiarlo sin necesidad de recompilar ningún código.
En tercer lugar, se crea al menos una clase de implementación con el código para acceder a la base de datos.

El patrón Factory es usado en el patrón DAO, para leer e instanciar la clase correspondiente que proporcione acceso a la base de datos.

El patrón Value List Iterator se usa para gestionar listas largas de datos. Este patrón permite al cliente pedir una página cada vez. Es aplicable a varias situaciones usuales, por ejemplo cuando el usuario sólo quiere acceder a una porción de la lista, o si una lista es demasiado grande para la memoria o la pantalla. También es útil para evitar transmisiones demasiado largas.

La estrategia empleada para implementarlo usa un bean de sesión con estado y otro sin estado. El bean sin estado accede a los datos y el bean con estado gestiona la iteración de la lista. El bean con estado presenta una interfaz prolija al cliente y mantiene el estado de los datos específicos del cliente, también traduce parámetros específicos del cliente a llamadas al

bean de sesión sin estado quien se encarga del acceso a los datos. Esta estrategia es la más eficiente implementación de este patrón dado que alivia a los clientes encapsulando los datos específicos del cliente en un bean con estado. Además, el bean sin estado requiere menos recursos del lado del servidor porque el contenedor EJB puede fácilmente intercambiar instancias entre clientes.

Es preferible siempre mantener los datos de una sesión en un bean con estado que hacerlo usando los objetos de sesión de las páginas JSP. A la larga, los datos en un bean con estado son mucho más fáciles de mantener que los datos en el componente web.

Un cliente local se ejecuta en la misma JVM que el bean al cual accede. Puede ser un componente web u otro EJB, pero no una aplicación cliente. Además, los parámetros de los argumentos son pasados por referencia en llamadas locales, esto puede ser arriesgado, dado que algún método puede usar esta referencia para inadvertidamente modificar el argumento que se le pasa (se le llama ‘efecto lateral’).

En la aplicación implementada, LlibreIteratorBean y CercarLlibre han sido diseñados para trabajar en tándem, lo cual hace que el último sea un buen candidato a para el acceso local y su ejecución en la misma JVM tiene sentido. Dado que los argumentos pasados requieren operaciones de sólo lectura, no habrá que preocuparse de los efectos laterales de los parámetros de los objetos.

Es posible implementar un bean con acceso local y remoto, aunque no es común en ambientes de producción, ya que no se puede acceder desde una aplicación a la interfaz local de un bean de empresa (lo cual es la forma típica de testear el código EJB). Se debe implementar el acceso remoto para el testing y la depuración.

Mediante del uso del patrón Session Façade se encapsulan llamadas al servidor que envuelven la ejecución de múltiples métodos de negocio
Se puede usar un bean de sesión con o sin estado y presenta las siguientes ventajas:

- El bean de sesión encapsula el proceso de negocio en un lugar y puede ser llamado por cualquier cliente. De manera que se simplifica el cliente.

- El bean de sesión accede al bean de entidad mediante llamadas locales, minimizando el número de llamadas remotas. Esto mejora el rendimiento.

- Usando Session Façade se aisla el bean de entidad de los clientes en general. Si se necesita cambiar la estructura de la base de datos, probablemente no se tenga que retocar nada del cliente. O sea que los cambios quedarían limitados al bean de entidad.

- Dado que los métodos del bean de sesión se ejecutan dentro de transacciones, se pueden ejecutar accesos múltiples al bean de entidad (los cuales pueden incluír actualizaciones de base de datos) fácilmente dentro

de una sóla transacción. De esta manera Session Façade simplifica operaciones que involucran múltiples pasos con un bean de entidad creando un único y transaccional método en el bean de sesión.

El patrón Composite View se utiliza para conseguir la visualización de una página mediante la composición de diversas. Esto evita la codificación repetitiva y aprovecha las clases ya compiladas por el servidor. En la implementación de la tienda se ha seguido un acercamiento similar, pero usando frames HTML. Dado que la única página que se actualiza es la central, no es necesario re-leer y compilar el resto de frames repetidamente.

MVC

La arquitectura MVC es una manera de dividir la funcionalidad entre los objetos involucrados en mantener y presentar datos de manera de minimizar la interdependencia entre ellos.
Model (Modelo), representa a los datos de la aplicación y las reglas de negocio que rigen el acceso y modificación de dichos datos.
View (Vista), despliega los contenidos del modelo. Específica cómo deben ser presentados los datos del modelo al cual accede.
Controller (Controlador), define el comportamiento de la aplicación, interpreta las acciones del usuario y las traslada al Modelo para que las ejecute.

Base de datos

Es una buena idea cerrar las conexiones a bases de datos en un bloque finally. Esta técnica asegura que la conexión se cerrará, incluso si se produce una excepción. Al cerrar la conexión, ésta retorna al pool (reserva) de conexiones del contenedor.

Siempre se debería acceder a la base de datos mediante la obtención de una conexión del contenedor EJB, Datasource. Esto permite que el contenedor asigne conexiones eficientemente desde un pool de conexiones. Dado que la obtención de la conexión no cuesta mucho en recursos, se puede adquirir en el momento de acceder a la base de datos y abandonarla luego de la consulta.

Siempre que sea posible es mejor usar la sentencia PreparedStatement de JDBC que Statement. Esto permite que la base de datos guarde la sentencia compilada internamente, y evita múltiple recompilaciones cuando el texto de la consulta permanece inalterado. De esta manera se incrementa el rendimiento, sobretodo necesario cuando hay muchas consultas de clientes.

Hay varias formas de enfrontar el problema de generar claves primarias únicas. Si se hace mediante programación usando bloques de claves es

eficiente. Un esquema que obtenga la siguiente clave primaria de la base de datos puede generar cuellos de botella debido a un uso elevado del sistema.

Respecto a las transacciones, se suelen llevar a cabo en una serie de pasos separados, por ejemplo:

```
try {
 startTransaction();
   reservaVuelo();
   pagarVuelo();
 commitTransaction();
} catch (Exception ex) {
   rollbackTransaction();
}
```

Para que esta transacción se complete, todos los pasos deben realizarse con éxito. De otra manera los datos grabados serían inconsistentes. Estos pasos separados forman el 'contexto de transacción', el cual es visto como una operación individual.

Una transacción 'commits' cuando todos los pasos se han realizado con éxito, por lo tanto se guardan los datos.

Una transacción 'rolls back' cuando uno o más de los pasos ha fallado y por lo tanto debe ser 'deshecho' cualquier paso anterior. De manera que los datos modificados vuelvan ha su estado original anterior al comienzo de la transacción. El primer paso que comienza la transacción y el paso que la lleva a cabo (commit) forman los límites de la transacción y se le llama la demarcación de la transacción.

El servidor de aplicaciones proporciona servicios de gestión de recursos para transacciones. Mediante la comunicación con la base de datos el gestor de transacciones puede bloquear registros involucrados en una transacción, de manera que otra consultas o actualizaciones quedan temporalmente bloqueadas.

Clases

Dado que el contenedor EJB controla los hilos (threads) de la aplicación, usar ArrayList es más eficiente que usar Vector, por ejemplo. A pesar de que ambas colecciones crecen automáticamente y son serializables, los métodos de búsqueda y almacenaje no están sincronizados en un ArrayList (sí en un Vector), de manera que no se duplica el proceso de sincronización. Además se pueden manipular los elementos fácilmente con Iterator.

Una declaración JSP comienza con <%! Y termina con %>. Se usan para definir variables y métodos a nivel de clase. Las variables declaradas aquí aparecen fuera del método _JspService(). Se inicializan al crearse la página JSP y tienen alcance de clase.

Implementación del web

Componentes:

Bean categoria- Se trata de un bean de sesión sin estado que se encarga de cargar las categorías dadas de alta en la base de datos. Su instancia se realiza a través de una declaración en el archivo categoria.jsp. Implementa el patrón DAO y utiliza el patrón Value Object para la comunicación con el servidor. Lanza la excepción a nivel de aplicación CategoriaDAOSysException.

Bean novedades- Se trata de un bean de sesión sin estado que se encarga de cargar las novedades dadas de alta en la base de datos. Su instancia se realiza a través de una declaración en el archivo dreta.jsp. Implementa el patrón DAO y utiliza el patrón Value Object para la comunicación con el servidor. Lanza la excepción a nivel de aplicación NovedadDAOSysException. Este bean tiene la función de ilustrar una funcionalidad que no implementa ya que sólo carga los primeros cuatro registros de la base de datos. En una aplicación real esto se puede implementar, o bien con una tabla complementaria en la base de datos, o bien leyendo las cuatro fechas más actuales de la tabla existente. El acercamiento depende del negocio.

Dada la función de ambos beans (se trata de una simple lectura de datos), el objeto VO que devuelve el bean lo lee directamente la página jsp. No se ha sobrecargado la implementación usando el modelo MVC en estos dos casos.

Bean cistella- Se trata de un bean de sesión con estado con una variable de instancia del tipo ArrayList principalmente. Se encarga de guardar los productos seleccionados por el usuario. Utiliza el patrón Value Object para la comunicación con el servidor. El funcionamiento de la cesta de la compra se ha implementado de la forma siguiente:
El usuario guarda un producto cada vez, no es posible, en un primer momento, guardar una cantidad mayor de un mismo producto. Para lograr una cantidad mayor que la unidad de un mismo producto se debe agregar dicho producto en repetidas ocasiones. Hay que tener en cuenta que se trata de una librería virtual, por tanto este comportamiento es coherente, dado que difílmente un usuario compra más de un ejemplar de un mismo libro. De esta manera se simplifican las acciones del usuario en el proceso de compra.
La cesta permite borrar un producto, si hay más de un ejemplar del mismo producto, borra uno cada vez. También permite borrar todo el contenido de la misma y hacer el pedido.
Se ha implementado usando el modelo MVC. El servlet CistellaServlet.java se encarga de gestionarla.

cercaIterator- Este componente está formado por dos beans, uno de sesión con estado que guarda el estado del cliente, y otro sin estado que se encarga de las operaciones con la base de datos. Implementa el patrón Value List Iterator ligeramente modificado (Value List Iterator Multiple), dado que pagina resultados de más de un tipo de búsqueda, guardando el estado de todas, enriqueciéndo así la navegación del usuario.
Los beans al respecto son LlibreIteratorBean y CercarLlibre que han sido diseñados para trabajar en tándem, lo cual hace que el último sea implementado para el acceso local y por tanto su ejecución es en la misma JVM.
También implementa el patrón DAO.
Lanza las excepciones a nivel de aplicación TitolNoTrobatException, AutorNoTrobatException y CercarDAOSysException y utiliza el patrón Value Object para la comunición de datos.
Se ha seguido el modelo MVC en la presentación y gestión de los datos. En este caso el servlet CercarServlet.java se encarga de ello.

guardarDades- Este componente no se ha implementado. La aplicación realizada tiene la finalidad de ilustrar el funcionamiento de una tienda virtual de forma eficiente simplemente. Así, por ejemplo, no se piden datos al cliente al realizar el pedido ni tampoco se guarda su selección. Por supuesto que en una aplicación real esto es imprescindible.
Para la implementación de este componente se pudo haber usado un CMP gestionado por un bean de sesión (Session Façade). Esta implementación nos impone la codificación en 'EJB Query Language' en el descriptor de despliegue.

Servlets

Se han implementado dos servlets que hacen el papel de controladores en el modelo MVC, CistellaServlet y CercarServlet.

JSPs

Su uso se limita a presentación de datos en combinación con HTML y JavaScript.

Implementación del gestor

Para el gestor de la tienda se ha seguido otro modelo, el que encierra la lógica de negocio en javabeans. La razón es que esta aplicación no experimentará un uso intensivo (en principio habrá un sólo administrador y las actualizaciones, aunque puedan ser frecuentes, requieren un tiempo mínimo), además los servicios que ofrecen los ejb's suelen ser muy costosos en recursos.

Implementa el patrón DAO, y utiliza el patrón Value Object para la comunicación con el servidor. Lanza la excepción a nivel de aplicación AdminDaoSysException.

En lo que respecta a la seguridad, se ha implementado seguridad básica contra la base de datos. Sólo tiene acceso el administrador (rol admin). Para ello se ha agregado un módulo en el fichero login-config.xml del servidor, al cual se hace referencia desde el descriptor de despliegue y se han creado las tablas correspondientes.

También implementa transacciones, en concreto, cuando se borra una categoría de la base de datos, se deben borrar todos los libros que engloba para no dejar registros huérfanos. Estos dos pasos se llevan a cabo mediante una transacción, de manera que si ambas actualizaciones no se realizan con éxito, no se realiza ninguna.

Notas:

La agrupación de datos, siempre que ha hecho falta, se ha implementado usando ArrayList. Esta clase resulta más eficiente dentro de esta arquitectura.

La aplicación permite dos criterios de búsqueda de productos, búsqueda por autor y búsqueda por título.

Todos las conexiones a base de datos se obtienen mediante el objeto Datasource, esto resulta más eficiente, y deja que el contenedor gestione las conexiones correctamente.

A excepción del bean que juega el papel de leer datos en la implementación del patrón Value List Iterator, todos los beans implementan la interfaz remota. Esto es debido a que la implementación de la tienda intenta ser lo más general posible y dejar las implementaciones locales para casos que ilustran su necesidad.

También se ha considerado que el número de unidades de un producto son ilimitadas, o sea, no se lleva un inventario al respecto, ni se consulta el número de unidades disponibles.

La aplicación presenta dos niveles de agrupación, categorias y subcategorias, en una aplicación real del mismo tipo podría haber más, de manera de conseguir una separación más fina y, facilitar la navegación y búsqueda por parte del cliente.

El gestor implementa la posibilidad de subir archivos (imágenes) al servidor y realiza una mínima gestión al respecto, se limita a subir el archivo en el caso de altas y borrarlo en bajas, no realiza comprobaciones de existencia de dicho archivo antes de las operaciones. La implementación de este proceso nos impone desplegar el archivo .ear y .war expandidos ya que Jboss no lo hace, o al menos no reconoce los paths de los archivos no expandidos.

CAPÍTULO 7

CONCLUSIÓN

Los objetivos que planteaba el proyecto en un primer momento eran los siguientes:

1. Seguimiento de un proceso de construcción de software (RUP)
2. Entender y describir los requerimientos de la aplicación usando diagramas UML.
3. Explorar la relación entre Java y UML.
4. Uso de java para crear aplicaciones del lado del servidor con JavaServer Pages, Servlets, javabeans y beans de empresa.
5. Implementación de una aplicación con componentes distribuídos.
6. Uso de patrones.
7. Eficiencia, rendimiento y portabilidad en la implementación
8. Familiarización con los métodos de seguridad en el acceso a aplicaciones.
9. Investigar los beneficios de desplegar aplicaciones Java en productos de código abierto.

Para conseguir los objetivos 1,2 y 3 se han repasado los módulos de la asignatura “Enginyeria de programari 1” y se han consultado varios libros. Se trataba de interpretar formalmente el proyecto y de ver la relación entre Java y UML, ambas cosas se han conseguido, en la medida que el alcance de proyecto lo exigía, lo cual ha permitido analizar lo que hacía falta y lo que sobraba respecto al proyecto que se planteaba.

El objetivo 4, quizá el más importante, se ha conseguido plenamente. Interpretar todos los conceptos que giran entorno a la plataforma J2EE eran complicado en sí mismo, el trabajo final revela el grado de fluidez en el manejo dichos conceptos.

El objetivo 5 planteaba el problema de trabajar con un servidor de código libre, JBoss y conseguir la traslación a la práctica de la teoría. Este objetivo ha permitido no sólo familiarizarse con cierta profundidad en dicho servidor sino terminar de encajar las diferentes piezas que forman el puzzle J2EE.

En el objetivo 6 se trataba de llegar a un grado de abstracción más profundo en el desarrollo de código. La idea de fondo es que una aplicación no sólo debe funcionar sino que debe preveer la eficiencia , el mantenimiento y la portabilidad. Se han usado a lo largo del proyecto varios patrones de software y se ha justificado su uso.

En el capítulo de implementación se detallan varios puntos referentes a la lógica en la utilización de los diferentes componentes y su eficiencia. Lo

cual forma el fundamento del objetivo 7, que se complementa con el anterior.

En el gestor se ha implementado seguridad básica contra una base de datos. Con esto se ha logrado ver de forma práctica los elementos que intervienen, en la aplicación y en el servidor, en el proceso de implementación de seguridad. Este era el objetivo 8.

Finalmente está el uso de tecnología de código abierto. En este punto se ha comprobado la fiabilidad y la buena documentación de estas aplicaciones.

Conclusión, no se puede hablar de objetivos no conseguidos o parcialmente conseguidos dado que se ha logrado implementar la idea que se tenía en un principio.

Mejoras

En cuanto a la tienda:

- La página que carga las “novedades” ilustra una funcionalidad que no implementa, es decir, aquí lo que hace es cargar los cuatro primeros registros de la tabla “tllibre”, cuando en una aplicación real debería cargar una tabla en concreto al respeto, o los registros con fechas más actuales.
- Lo mismo pasa con el banner. El contenido del banner debería ser leído de la BD, y, actualizado en el gestor.
- El cuadro que informa del web (“quienes somos”, “Cómo comprar”..) no es funcional.
- No se piden los datos al cliente ni tampoco se guarda el pedido realizado en la BD. Para hacerlo se pudo haber usado CMPs juntamente con el patrón Session Façade.
- En caso de haberse implementado el punto anterior se usaría SSL para la transmisión del pedido y los datos del cliente, o algún mecanismo similar.
- En cuanto al código, no se han centralizado en una clase las llamadas a los diferentes beans.
- También se ha considerado que el número de unidades de un producto son ilimitadas, o sea, no se lleva un inventario al respecto, ni se consulta el número de unidades disponibles.
- La aplicación presenta dos niveles de agrupación, categorias y subcategorias, en una aplicación real del mismo tipo podría haber más, de manera de conseguir una separación más fina y, facilitar la navegación y búsqueda por parte del cliente.

En cuanto al gestor:

- El gestor implementa seguridad básica, lo cual implica la transmisión de los datos de login en texto claro. Este grado de seguridad en una aplicación real es totalmente insuficiente.

- Desde el gestor se deberían poder leer los pedidos y confirmar los envíos, así como gestionar su tramitación.
- También se debería poder dar de alta otros usuarios.
- El gestor implementa la posibilidad de subir archivos (imágenes) al servidor y realiza una mínima gestión al respecto, se limita a subir el archivo en el caso de altas y borrarlo en bajas, no realiza comprobaciones de existencia de dicho archivo antes de las operaciones.

Bibliografía

[1] ANDERSON, G., ANDERSON, P. Enterprise JavaBeans™ Component Architecture: Designing and Coding Enterprise Applications. (Marzo 2002).

[2] AHMED, K., UMRYSH, C. Developing Enterprise Java Applications with J2EE™ and UML. (Octubre 2001).

[3] REED JR, P. Developing Applications with Java™ and UML. (Noviembre 2001).

[4] EELES, P., HOUSTON, K., KOZACZYNSKY, W. Building J2EE™ Applications with the Rational Unified Process. (Agosto 2002).

[5] METSKER, S. Design Patterns Java™ Workbook. (Marzo 2002).

[6] MONSON-HAEFEL, R. Enterprise JavaBeans, 3rd Edition. (Setiembre 2001).

[7] ALLAMARAJU, S., BEUST, C., DAVIES, J. Programación Java Server con J2EE Edición 1.3. (Febrero 2001).

[8] HALL, M. Servlets and JavaServer Pages™. (Mayo 2000).

[9] HALL, M. More Servlets and JavaServer Pages™. (Diciembre 2001).

[10] TAYLOR, L., THE BOSS GROUP. Getting Started with Jboss

[11] JBoss 3.0 documentation. (web).

[12] Out-of-the-Box 1.0 by EJB Solutions. (web).

[13] STARK, S., FLEURY, M., THE BOSS GROUP. JBoss™ Administration and Development. (Marzo 2002).