Proyecto TFC	**eJugueteLandia**
Documento	**MEMORIA**
Autor	Javier Martín Rubio EITG
Consultor	Jordi Ceballos
Versión – Fecha	1.0 – 20/06/2004

1 Dedicatoria y reconocimiento

Mi agradecimiento a todas aquellas personas interesadas en el estudio de este trabajo y sus conclusiones, y muy especialmente a las personas que me quieren, por haber aceptado y entendido que no les pudiera dedicar mucho tiempo.

2 Resumen

Este proyecto, a diferencia de lo que creo suele esperarse de un trabajo de estas características, tiene tres objetivos básicos; el primero, permitir a cualquier persona que disponga de conexión a Internet, la consulta y adquisición de los productos catalogados por nuestro cliente ficticio, *JUGUETELANDIA, S.A*, limitando dicha oferta a juguetes y productos relacionados, pero manteniendo la posibilidad de tratar cualquier tipo de artículo; segundo, entregar a nuestro cliente un producto de calidad y en el que se garantiza la extensibilidad y reutilización del modelo, si bien es cierto que debido a las restricciones temporales impuestas no ha sido posible aplicar ciertas mejoras; y por último, proporcionar un sistema intuitivo y de sencillo manejo que facilite la interacción hombre-máquina.

Aparte de lo ya mencioando, también se ofrecen una serie de funcionalidades -ya implementadas- que facilitan la consulta y mantenimiento del catálogo, el registro de visitas de los navegantes y, cómo no, un registro de usuarios; siendo éstos últimos, junto con la autenticación ante el sistema, requisitos imprescindibles para poder tramitar algún pedido.

Todas estas funcionalidades han sido desarrolladas mediante tecnologías de vanguardia, como la plataforma J2EE de Sun Microsystems y Jakarta Tomcat de Apache, todos ellos de reconocido prestigio en la comunidad OpenSource, sin olvidar todas aquellas que hacen de este proyecto una solución profesional, como XHTML o JavaScript.

3 Índice de contenidos

4 Cuerpo de la memoria

4.1 Capítulo 1: Introducción

4.1.1 Justificación del proyecto

En el mercado pueden encontrarse diferentes aplicativos diseñados para el comercio a través de la red, pero aunque dispongan de un gran número de funcionalidades, distan mucho de ser la solución ideal, ya que no tratan convenientemente ciertos aspectos, algunos de ellos de importancia capital: el diseño de las ventanas y la navegación a través de las diferentes opciones.

Con este proyecto no se persigue crear un aplicativo de apariencia espectacular y tampoco competir con aplicativos de similares características, aunque sí ofrecer comodidad, sencillez y facilidad en el manejo de un aplicativo web, sin olvidar todos aquellos aspectos técnicos que facilitarán el mantenimiento y futura ampliación. Aunque de esto último no suelen ser conscientes los navegantes, la estabilidad y robustez de la solución repercutirá positivamente en la imagen corporativa de la empresa.

4.1.2 Objetivos del proyecto

Los objetivos que se persiguen son de diversa índole, pero básicamente y para resumir, se agruparán de la siguiente manera: objetivos de software y objetivos técnicos. El primero hace referencia a todos aquellos requisitos o funcionalidades que se le exigen al aplicativo; el segundo, a las cuestiones técnicas o de diseño que influyen directamente sobre los primeros.

4.1.2.1 Objetivos de software

Desarrollar un aplicativo de comercio electrónico que pueda ser utilizado desde cualquier punto de Internet y por cualquier persona, incluyendo todas aquellas con un cierto grado de minusvalía visual.

Entre todas las funcionalidades, destacaremos las siguientes:

- Consulta del catálogo.

  Mostrar una lista completa productos y ordenada según las necesidades del usuario, pero en cualquier caso, esta ordenación estará limitada a la referencia o fecha de inserción en el catálogo.

- Realizar búsquedas en el catálogo en base a diferentes criterios.

  Estas búsquedas podrán realizarse por edad, precio y categoría.

- Consultar la ficha de un determinado producto.

  Visualizar toda la información disponible del producto, mostrando la imagen de éste si se encuentra disponible. Si el usuario se ha registrado y autenticado, se permitirá añadir el producto al carrito. Esta consulta podrá solicitarse desde cualquier ventana que muestre una lista de productos.

- Registro de usuarios.

  Todo navegante que esté interesado en adquirir algún producto deberá introducir en un formulario sus datos personales, así como un identificador y clave de acceso.

- Adquirir los productos del catálogo y modificar la composición del carrito.

  Las operaciones sobre el carrito se limitan a la consulta, recálculo, confirmación, borrado y modificación del número de unidades compradas. Tanto la adquisición como las operaciones descritas podrán realizarse únicamente por usuarios registrados.

- Registro de visitas y consulta de las mismas.

  Se requerirá de algún tipo de mecanismo que permita a *JUGUETELANDIA* conocer las opiniones de sus potenciales clientes, así como visualizar la lista completa de sugerencias o comentarios realizados.

- Mantenimiento completo de productos.

  Ampliar o modificar la composición del catálogo: altas, bajas o modificaciones. Esto solamente podrá ser realizado por el administrador del sistema.

- Ofrecer al usuario información de la empresa, entre otros, la ubicación y servicios prestados.

Respecto a las vistas de usuario, también se impone la necesidad de que éstas sean intuitivas y permitan una cómoda navegación.

4.1.2.2 Objetivos técnicos

- Facilitar la extensibilidad y reutilización del modelo o de ciertas partes del mismo.

- Evaluar y aplicar tecnologías de vanguardia y de reconocido prestigio, con especial intereses por las de fuente libre (Open Source).

- Facilitar la incorporación de mejoras y ampliaciones en futuras fases del proyecto.

4.1.3 Enfoque y método seguido

4.1.3.1 Comentarios previos

Este proyecto fue concebido para dar respuesta a las necesidades de un cliente imaginario, *JUGUETELANDIA.* Éstas consistían en facilitar a cualquier persona que dispusiera de conexión a Internet la consulta y adquisición de juguetes, con posibilidad de ampliar la oferta a otro tipo de productos, si así se requiriese.

Realizado el estudio y análisis del problema, así como la planificación temporal de todo el desarrollo, se concluyó que no era posible ofrecer un producto con la calidad deseada en el plazo de tiempo establecido, aunque se consideró interesante continuar con el proyecto, ya que ello permitiría obtener un mayor conocimiento de la tecnología existente, así como dar a conocer nuestra filosofía de trabajo y señas de identidad.

La calidad de un producto lo es todo, aunque de ello, quizás, algunos desarrolladores no sean del todo conscientes. El producto resultante es de una gran calidad, pero ello no implica que se hayan tomado siempre las soluciones idóneas, ya que para ello -tal y como se ha indicado- hubiera sido necesario ampliar la temporalización de las tareas planificadas.

En este proyecto se ha intentado plasmar el conocimiento adquirido de varios años, tanto a nivel de ingeniería de programación como de diseño de bases de datos, sin olvidar ni las vistas de usuario ni a las personas que las deben utilizar. Esto último es básico si creemos realmente en la necesidad de que todo el mundo pueda utilizar nuestros desarrollos, incluso aquellos que pudieran presentar alguna minusvalía.

4.1.3.2 Enfoque

La aplicación **eJugueteLandia** ha sido desarrollada con la noción de una arquitectura de 3 capas: presentación, empresa y datos. Los motivos de esta decisión son muchos, pero en suma, este enfoque nos facilitará la incorporación de nuevas funcionalidades y/o la revisión de las existentes, así como también la reutilización de parte del modelo.

En el caso de **eJuguetelandia**, creo evidente, en este caso, la necesidad de disponer de un cliente ligero (Thin Client), ya que entre sus potenciales usuarios tendremos a cualquier posible navegante, presuponiendo -aunque realmente no sea así- que lo único en común entre sus sistemas informáticos sea la mera existencia de un explorador.

Como este proyecto podría ser ampliado con nuevas funcionalidades, algunas de ellas –posiblemente- no orientadas al mundo de Internet, y siempre deseando no descartar otras posibilidades, como los aplicativos cliente/servidor mucho más adecuados en determinadas ocasiones-, se hace recomendable un lenguaje de programación multipropósito que pueda resolver cualquier tipo de problemática y orientación del aplicativo.

El lenguaje de programación utilizado será Java y la plataforma J2EE (Ver Software). Se ha escogido esta tecnología, aparte de lo ya comentado, porque nos permite implementar nuestra solución informática en diferentes sistemas, disponer de objetos gestionados, reutilización y modularidad, así como poder crear aplicaciones de servidor. Entre las extensiones que serán utilizadas podemos destacar las siguientes: JSP, Servlets, y JDBC.

Aunque hubiese sido interesante la adopción de los EJB, ya que nos ofrecen un mayor nivel de independencia, la planificación lo ha impedido, pero el diseño del aplicativo facilita que puedan ser incorporados en próximas fases.

También se había pensado utilizar protocolos de seguridad en las comunicaciones, como SSL, pero no ha sido posible debido a lo ya mencionado anteriormente.

4.1.3.3 Metodología

Se puede encontrar más información sobre este tema posteriormente en el Documento de Especificaciones.

Se ha seguido un ciclo de vida clásico y del que, a modo de resumen, se mencionan las siguientes actividades:

- Obtención de un documento de especificación con los requisitos más importantes del aplicativo a desarrollar y proponer en el proyecto.
- Planificación del proyecto.
- Analizar los requisitos: identificación de subsistemas, análisis y diseño de las vistas de usuario; y el estudio, extensibilidad y confección de los diagramas de casos de uso.
- Diseño del aplicativo: representación gráfica de los subsistemas y sus relaciones, identificación de clases, así como también responsabilidades y colaboraciones entre objetos, confeccionar diagramas UML, estudio de reutilización, definir de la base de datos.
- Implementación:

  1º. Crear la estructura de la BD y juego de pruebas mediante diferentes scripts SQL.

  2º. Crear una estructura de carpetas acorde con las necesidades.

  3º. Implementar clases genéricas y realizar pruebas individuales.

  4º. Implementar las entidades propias del sistema.

  5º. Implementar los gestores de conexión y de acceso (DAO) y realizar pruebas individuales.

  6º. Implementar los servlets generales de la aplicación y realizar pruebas individuales.

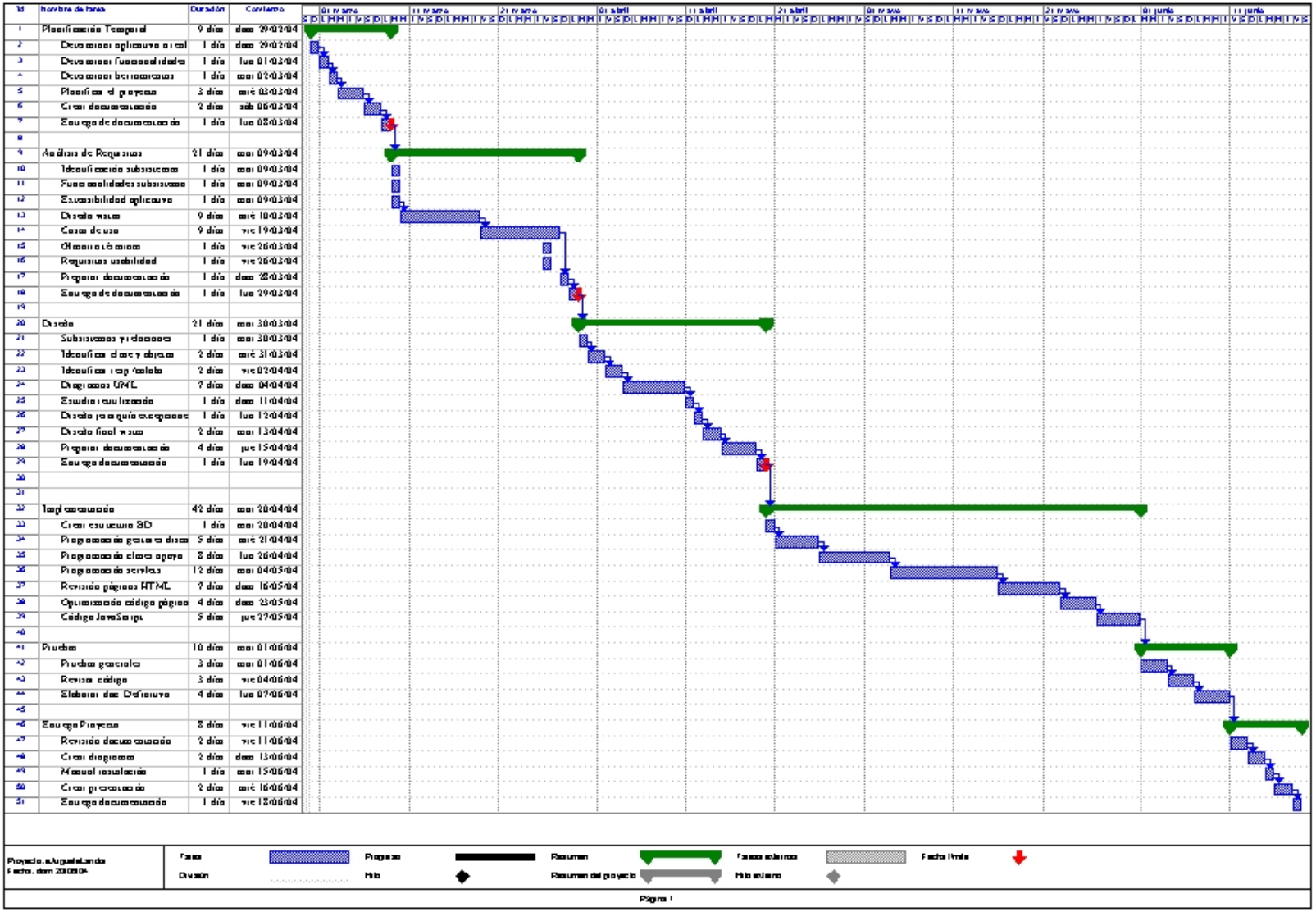
  7º. Crear páginas externas de estilos CSS y archivos externos JavaScript con funciones generales.

  8º. Subsistema 1-7: optimización de las páginas JSP, implementación de los servlets propios del subsistema, añadir el código JavaScript necesario y realizar pruebas generales.

  9º. Pruebas entre subsistemas.

- Pruebas de calidad.
- Preparar documentación y revisión de los documentos de diseño.

4.1.4 Planificación del proyecto



4.1.5 Productos finales obtenidos

El producto resultado de este proyecto es *JUGUETELANDIA*, cuyas funcionalidades han sido repartidas entre diversos subsistemas, todos ellos independientes y con posibilidad de ser ampliados o entregados por separado.

4.1.5.1 Subsistema 1: Búsquedas y consultas de catálogo

Estarán incluidas en este subsistema todas aquellas opciones que tengan por objetivo mostrar una lista con los productos ofertados, con posibilidad de limitar u ordenar la salida por diferentes criterios. También se permite consultar toda la información de un determinado producto y, si el usuario se ha autenticado, añadir éste al pedido.

4.1.5.2 Subsistema 2: Compras

Bajo este título se engloban todas aquellas opciones reservadas para usuarios registrados y relacionadas con la adquisición de productos, tales como operar con el pedido en curso.

4.1.5.3 Subsistema 3: Visitas

Se permite a todos los usuarios que lo deseen, registrados o no, firmar en el libro de visitas y consultar las visitas realizadas.

4.1.5.4 Subsistema 4: Mantenimientos

Permite realizar un mantenimiento completo de productos: altas, bajas y modificaciones del catálogo.

4.1.5.5 Subsistema 5: Información

Incluye una serie de páginas estáticas con información diversa: horarios, tarifas, localización y emplazamiento de las oficinas.

4.1.5.6 Subsistema 6: Conexión

Permite la conexión y desconexión del sistema a usuarios ya registrados, así como también el crear cuentas de cliente a todos los navegantes interesados en la adquisición de productos.

4.1.6 Descripción del resto de capítulos

En el resto de capítulos que contiene esta memoria el lector podrá profundizar en el proceso de elaboración de este trabajo, así como conocer los motivos que han llevado a la toma de ciertas decisiones y a valorar en su justa medida las conclusiones alcanzadas, todas ellas basadas en la experiencia acumulada, no ya solamente la que me ha proporcionado el propio estudio y confección de este proyecto, sino también la experimentada en mi carrera profesional de casi ya veinte años..

En el capítulo 2 se nombran y describen brevemente las características del software utilizado durante el proyecto, así como la plataforma recomendada para soportar el aplicativo.

En el capítulo 3 se hace un repaso general de aquellos aspectos que considero más interesantes del aplicativo desarrollado, justificando las decisiones adoptadas (si se considerase que no resultan evidentes).

En el capítulo 4 se introducen posibles líneas de trabajo futuras, tanto las posibles revisiones del aplicativo existente como la ampliación de funcionalidades. En el caso de las revisiones, se realiza una valoración crítica de la solución aportada y se proponen las soluciones que se consideran más apropiadas.

En el capítulo 5 se presentan toda la serie de conclusiones que se desprenden del proyecto desarrollado, así como la conveniencia de aplicar ciertos procedimientos metodológicos.

En anexos, se encontrará un apartado que personalmente considero muy interesante y que está íntimamente relacionado con el diseño de las vistas de usuario.

También se ha confeccionado un documento de pruebas, pero no se ha considerado oportuno adjuntarlo a esta memoria, por este motivo se entregará como un documento aparte, junto con el documento de diseño actualizado con los cambios producidos en la fase de implementación.

4.2 Capítulo 2: Estudio de software

4.2.1 Plataforma tecnológica

El producto final de este desarrollo puede ser desplegado sobre cualquier plataforma existente, entendiendo por plataforma todos aquellos recursos tecnológicos, tanto físicos como de software, que hacen operativa una solución.

No es objetivo de este proyecto evaluar aspectos tecnológicos físicos, como servidores u otro hardware, aunque sí todos aquellos que tienen una relación mucho más estrecha con la solución aportada, tales como servidores y/o contenedores web.

4.2.1.1 Servidores web

En el mercado podemos encontrar diferentes servidores Web, y aunque cada uno de ellos tiene sus características específicas, suelen proporcionar los mismos servicios básicos. En nuestro caso, no tenemos dudas a la hora de escoger a Apache como primer candidato, entre otros motivos, porque está ampliamente extendido y reconocido por la comunidad Open Source.

4.2.1.2 Tecnologías para servir contenido dinámico

Podemos encontrar diferentes tecnologías: CGI, ASP, PHP, Perl, Servlet/JSP, todas con sus ventajas e inconvenientes, pero nos decantamos en todo caso por los servlet/JSP; en primer lugar, porque éstos se programan en Java, un lenguaje de propósito general ampliamente extendido, segundo, porque en comparación, por ejemplo, con CGI o ASP, ofrecen un mejor rendimiento, ya que se cargan una sola vez.

4.2.1.3 Contenedores Web

Las decisiones tomadas en los puntos anteriores señalan claramente a Jakarta Tomcat como contenedor web, ya que puede ser incorporado en un servidor Apache y soporta los componentes web mencionados: servlets y JSP,s.

Producto	Nombre y versión
Contenedor Web	Jakarta Tomcat 4.1.30

4.2.1.4 Contenedores EJB

En este proyecto no se requerirá de ninguno, ya que se ha previsto utilizar componentes EJB, aunque podrían ser incorporados en próximas ediciones del proyecto.

4.2.1.5 Plataforma Java

Todo lo anteriormente indicado nos conduce a la inexorable y conveniente aceptación de la Plataforma Java 2, Enterprise Edition (J2EE) como la más idónea para desarrollar este aplicativo.

4.2.2 Software complementario

Aunque ya ha quedado establecido cuál será la tecnología a utilizar, quedan también por concretar otros temas, como el de la persistencia. Es necesario almacenar y recuperar datos, por consiguiente se requiere de algún sistema que nos permita realizar dichas acciones: software motor de base de datos y el driver correspondiente para acceder desde aplicativos Java.

La elección en este caso, aunque importante, no tiene repercusiones serias en la extensibilidad, ampliación y mantenimiento del aplicativo, ya que las clases implementadas acceden a la bases de datos usando sentencias SQL estándar.

La elección ha sido MySQL, por su simplicidad y sencillez, mas si debieran ampliarse ciertas funcionalidades y fuera necesario que la propia BD gestionara algunos eventos, entonces aconsejaría Prosgress o FireBird, ya que ambas también son OpenSource, pero disponen de las características propias de un potente gestor de base de datos, entre otras, disparadores y procedimientos almacenados.

Producto	Nombre y versión
Motor de base de Datos	MySql Database Server 4.1
Driver MySql para Java	MySql Connector/J 3.0
Gestión de la base de datos	MySql Control Center 0.9.4

4.2.3 Herramientas de apoyo

En todo proyecto de estas características se hace recomendable simplificar ciertas tareas y esto se consigue con la inestimable ayuda de herramientas de apoyo.

Aunque algunas de ellas pueden ser utilizadas para diferentes propósitos, en la mayoría de los casos su uso se limita a tareas específicas que, en suma, serían las siguientes: documentación y presentación de contenidos, planificación, diseño, programación y pruebas.

4.2.3.1 Documentación y presentación de contenidos

- **MS Word 2000**

  Procesador de textos ampliamente extendido y de uso corriente que facilita todas aquellas tareas en las que resulte necesaria la redacción y proceso de documentos.

- **MS PowerPoint 2000**

  Al igual que el anterior, éste también es de uso habitual, facilitando la creación de presentaciones.

- **CutePDF 1.1**

  Convierte cualquier tipo de documento a formato PDF, formato éste mucho más adecuado para las entregas definitivas, ya que no permite cambios y es uno de los más extendidos.

4.2.3.2 Planificación

- **MS FrontPage 2000**

  Facilita la confección de documentos de planificación.

4.2.3.3 Diseño

- **MagicDraw 7.0**

  Potente herramienta que permite crear todo tipo de diagramas y puede ser integrado con NetBeans.

4.2.3.4 Programación

- **NetBeans 3.6**

  Entorno de desarrollo muy completo y que permite implementar cualquier tipo de componente de Java, ya sean clases individuales, servlets , EJB o JSP, entre otros. Incorpora un motor TOMCAT que facilita las tareas de comprobación y chequeo del aplicativo, así como toda una serie de utilidades para dar formato al código y simplificar ciertas operaciones.

  Este aplicativo es gratuito y está disponible para entornos Windows o Linux.

- **MS FrontPage 2000**

  Simplifica la creación de páginas HTML, pero no dispone de herramientas para la optimización y verificación de código, así como para la conversión automática a otros lenguajes de marcas, como XHTML.

  Aunque permite tratar páginas de estilos, no incorpora automatismos para su creación, aunque sí ventanas de diálogo que permiten indicar las propiedades de los elementos. El código CSS resultante no es compatible al 100% con las especificaciones de la W3C.

4.2.3.5 Pruebas

- **Jstyle 4.6**

  Software que proporciona una serie de métricas OO (MPC, PAH, NDD, ACO, RPC y CCM entre otras), permitiendo estimar la calidad del diseño resultante.

4.3 Capítulo 3: Aspectos relevantes

4.3.1.1 Tratamiento uniforme de las entidades

Aunque cada una de las entidades objetos del dominio son diferentes entre sí, también tienen ciertas características comunes, tales como un código y una descripción que las identifican inequívocamente; ello permite y facilita un tratamiento uniforme de éstas en todos aquellos procesos en las que intervengan. Un ejemplo práctico de su utilidad sería la creación de listas de objetos, comentado en el siguiente apartado.

El primer paso, por lo tanto, ha sido crear una jerarquía de clases; en el primer nivel se encontraría la superclase, abstracta y que encapsula los atributos y métodos de cualquier elemento; en el último, las entidades objetos del dominio, todas ellas instanciables y con sus atributos y métodos específicos.

4.3.1.2 Implementación de listas de objetos

En cualquier tipo de desarrollo se hace imprescindible disponer de elementos que nos permitan crear listas de objetos, sin importar a qué tipo concreto correspondan. La clase Vector, del paquete java.util, permite disponer de estas listas, pero se trata de una clase de uso genérico que, aún cumpliendo su función, es insuficiente para un proyecto de esta envergadura.

Como se ha tenido desde un buen principio la precaución de crear una apropiada jerarquía de clases para tratar las entidades, la implementación de listas específicas no supone pues un gran esfuerzo de implementación, pero aunque así hubiera sido, habría merecido la pena, ya que los beneficios posteriores son muchos y evidentes.

4.3.1.3 Menú de opciones dinámico

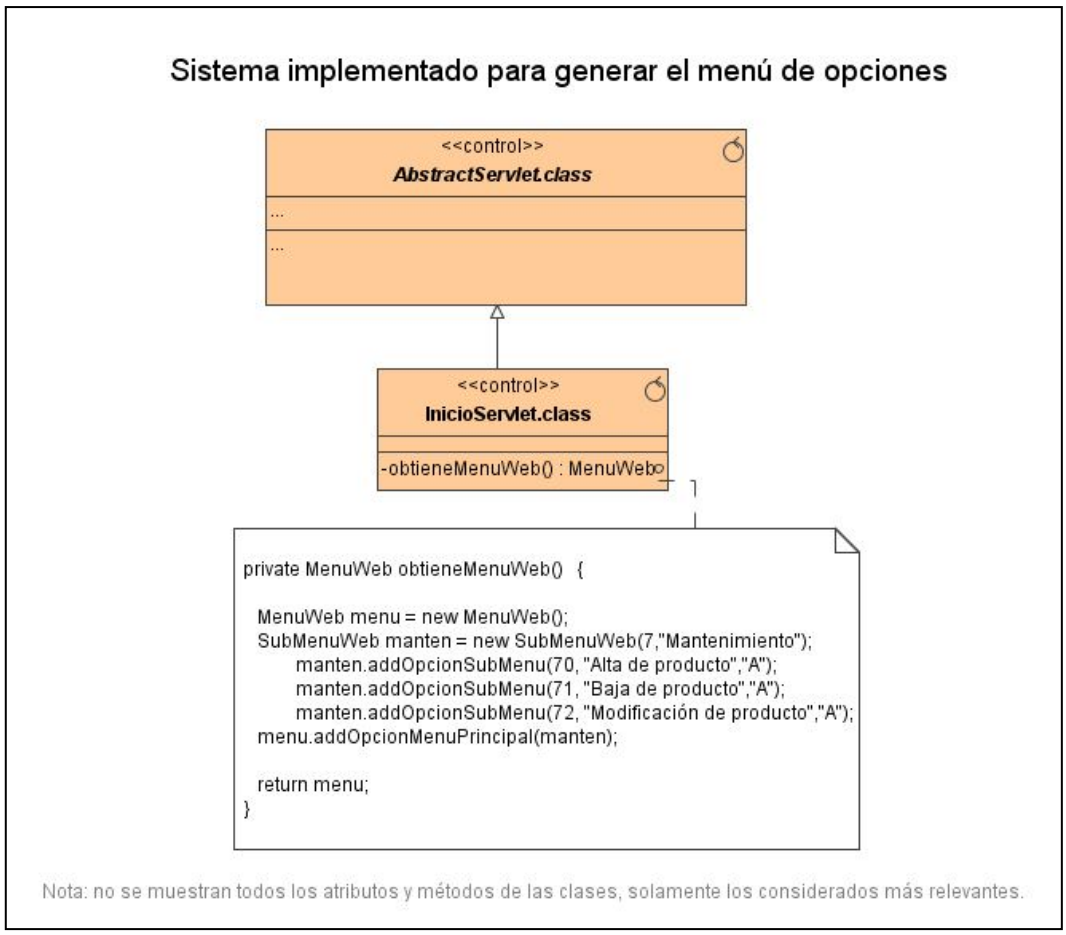
Aunque este aplicativo no dispone de excesivas funcionalidades, siempre resulta conveniente estructurar los contenidos de las ventanas del usuario. Esta estructuración comporta la utilización de algún tipo de mecanismo que facilite la tarea de visualizar las diferentes opciones en función de la tarea que se pretende realizar; de esto se encargará nuestro menú de opciones.

Siguiendo la línea marcada desde un principio, resultaba del todo impensable simplificar la cuestión del menú a un conjunto de elementos estáticos, ya que ello podría, en un futuro, dificultar tareas de mantenimiento y actualización. Además, también se hacía necesario resolver la problemática del acceso a las diferentes opciones en función del perfil de usuario.

La solución más razonable pasaba por la implementación de una serie de clases genéricas que nos facilitaran la tarea de gestionar cualquier tipo de menú, y no únicamente el correspondiente a éste desarrollo.

En esta primera versión del aplicativo y debido a los restricciones temporales impuestas, el menú de opciones se genera en un servlet, aunque lo más adecuado sería obtener las diferentes opciones de una base de datos, pero ésto ya sería tratado en futuras fases.

A continuación se muestra mediante una imagen, y sin entrar en ulteriores detalles, cómo se ha tratado este apartado y las clases que intervienen en el proceso.



(Figura 1)

4.3.1.4 Adopción de patrones de diseño

El aplicativo **eJugueteLandia** ha sido implementado siguiendo los principios de una serie de técnicas que tienen como objetivo la extensibilidad y reutilización del modelo, entre ellas, la aplicación de patrones de diseño. La adopción de algunos de éstos y la arquitectura Servlet-Centric-Design son parte de una vía hacia la arquitectura MVC (Modelo-Vista-Controlador).

De entre todos los patrones de diseño existentes se han adoptado los que a continuación se mencionan:

- Controlador frontal.

  Todas las páginas JSP son llamadas desde un servlet, siendo éste el que ejerce las funciones de controlador frontal.

- Vista composición.

  La ventana principal de usuario se ha dividido en múltiples sub-vistas y que, al ser recombinadas, muestran la vista global requerida.

- Objeto de acceso a datos.

  Las clases DAO construidas encapsulan el acceso a las diferentes tablas de las que se compone el aplicativo. Con esto se persigue independizar la lógica de empresa y presentación de la fuente de datos.

- Ayudante de vista.

  Se evita en la manera de lo posible incluir demasiado código Java en las diferentes páginas y esto se consigue con los Beans de apoyo implementados.

- Vista lanzador.

  Se combinan varios de los patrones aquí mencionados, tales como el Controlador frontal y Ayudante de vista; entre otros objetivos, para separar el flujo de trabajo de la presentación. Esto lo podemos encontrar, por ejemplo, en la página que permite operar con el carrito, ya que todas las posibles peticiones del usuario son tratadas por un único servlet, responsable éste de enviar la petición al servlet o página JSP correspondiente.

- Manejador de lista de valores.

  Los datos son almacenados en la memoria caché y son recuperados al ser solicitados.

4.3.1.5 Entidades del sistema vs. Beans

Como ya se ha comentado, los Beans permiten simplificar las páginas JSP y ayudan a conseguir una mayor separación entre la lógica de empresa y la presentación, pero ¿qué diferencia existe entre un Bean y una entidad cualquiera?. En principio, los Beans son clases corrientes que tendrán que cumplir con una serie de reglas:

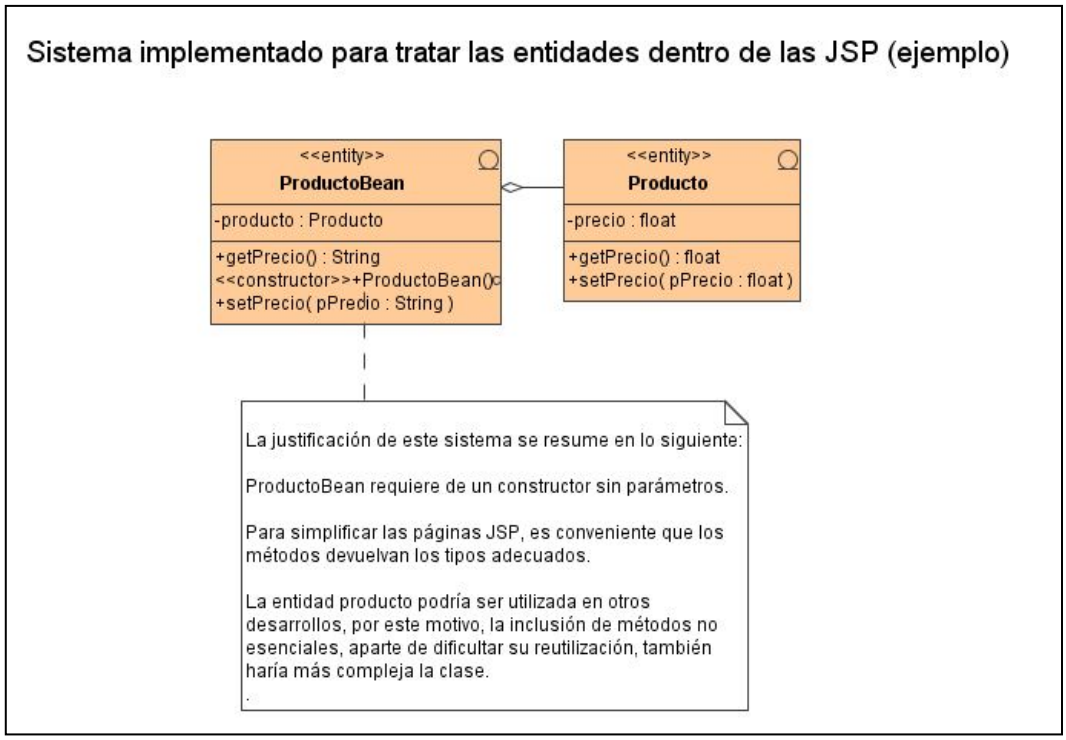
- Debe existir un método constructor sin parámetros.
- Por cada atributo, deben haberse definido los métodos getter y setter correspondientes.

En un principio, y con los primeros desarrollos, se utilizaron las entidades propias del sistema como si de Beans se tratara, ya que, a excepción del constructor sin parámetros, todas ellas disponían de los métodos anteriormente indicados. A medida que avanzaba el proyecto se llegó a la conclusión que esto había sido una mala decisión, ya que acarreaba una serie de importantes inconvenientes:

- Al existir un constructor sin parámetros, algo que no estaba previsto en el diseño original, se producían una serie de respuestas y/o excepciones inesperadas y de complicada detección.

- Con objeto de evitar incluir en las páginas JSP el código Java de conversión entre tipos de datos –especialmente los numéricos y fechas – y capturar las posibles excepciones, parecía razonable plantearse la modificación de los métodos de las entidades para que devolvieran o aceptaran los tipos de datos apropiados, pero esto comportaba un problema: sobrecargar la entidad con métodos que solamente eran utilizados por las JSP.

La solución adoptada, que supone una mejora importante para la estabilidad, robustez, reutilización y futuro mantenimiento, es la creación de toda una serie de Beans sin atributos propios – a excepción del que contiene la entidad objeto de tratamiento y que es pasado al Bean mediante un constructor específico- y en el que se han definido los métodos que requieren las JSP.


Sistema implementado para tratar las entidades dentro de las JSP (ejemplo)
<<entity>>
ProductoBean
-producto : Producto
+getPrecio() : String
<<constructor>>+ProductoBean()
+setPrecio(pPrecio : String)
<<entity>>
Producto
-precio : float
+getPrecio() : float
+setPrecio(pPrecio : float)
La justificación de este sistema se resume en lo siguiente:
ProductoBean requiere de un constructor sin parámetros.
Para simplificar las páginas JSP, es conveniente que los métodos devuelvan los tipos adecuados.
La entidad producto podría ser utilizada en otros desarrollos, por este motivo, la inclusión de métodos no esenciales, aparte de dificultar su reutilización, también haría más compleja la clase.

(Figura 2)

4.3.1.6 Jerarquía de Servlets

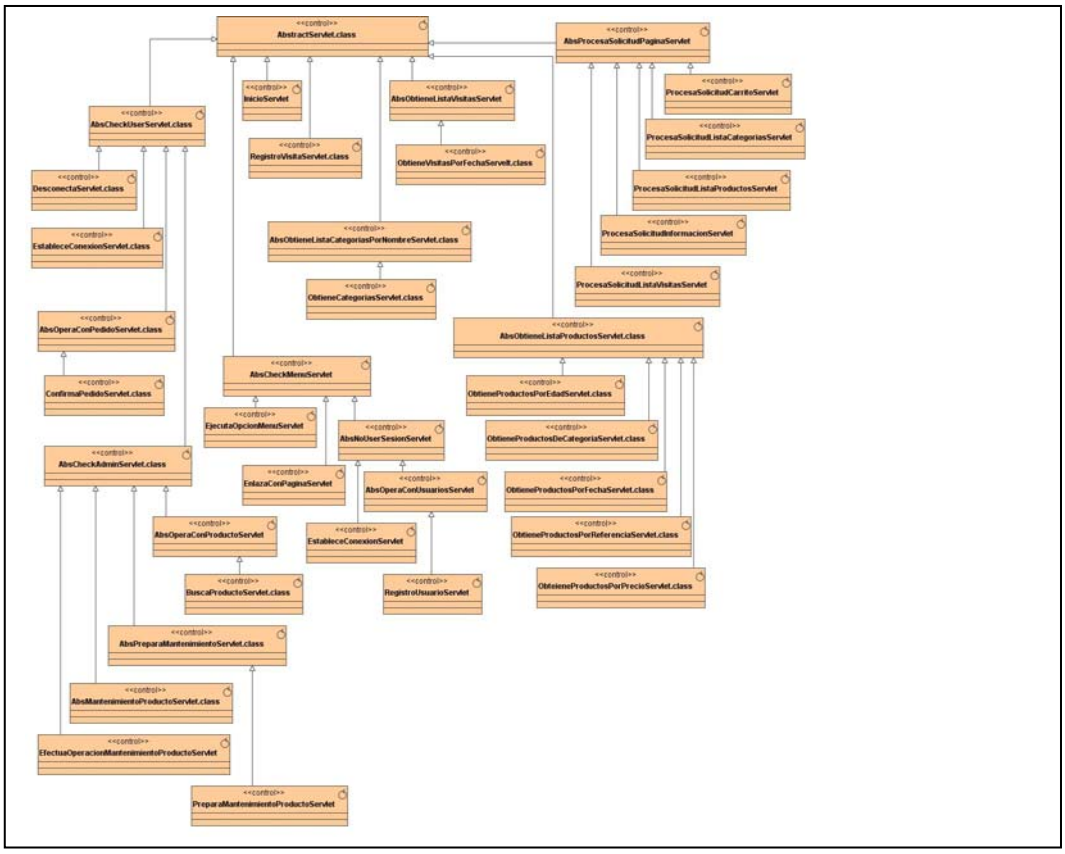
Como con cualquier otro componente de una aplicación debemos cerciorarnos de que no se producirán anomalías de funcionamiento debido o ciertas inconsistencias, como por ejemplo la no existencia de un determinado valor, cuando en teoría, siguiendo el flujo normal de proceso, éste debería existir.

Aunque esto resulta ciertamente más improbable con otro tipo de aplicativos, en los entornos web, un usuario que disponga de la barra de ¿direccionamiento? habilitada en su explorador podría introducir la url completa de un recurso del sistema, ya sea una JSP o servlet. Por este motivo, una de las medidas adoptadas ha sido la de desactivar dicha barra, aunque no la única, ya que no siempre es posible o conveniente mantener esta restricción.

Como los aplicativos deben reaccionar adecuadamente ante cualquier suceso inesperado y, a su vez, mantener optimizados los procesos de verificación, la solución adoptada ha pasado por la implementación de una jerarquía de servlets, responsables la mayoría de ellos de realizar las comprobaciones pertinentes.

Esto comporta una gran ventaja: en los servlets de nivel inferior (no superclases) podemos concentrar todos nuestros esfuerzos en implementar los procesos o tareas específicas de éstos, ya que las comprobaciones son responsabilidad de las clases superiores.

Este planteamiento también simplifica notablemente la incorporación de nuevas comprobaciones, la reutilización del modelo y por tanto su futuro mantenimiento.

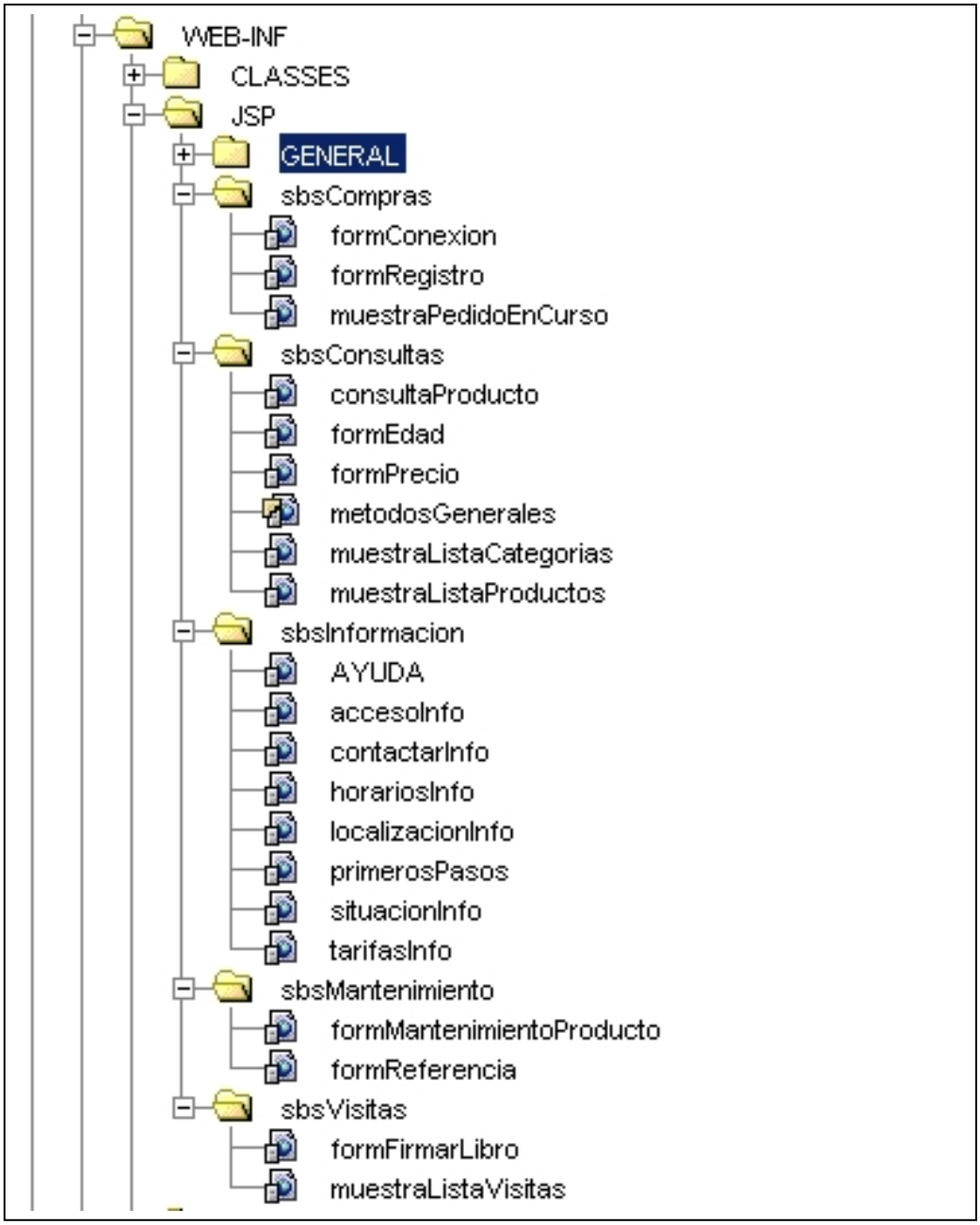


(Figura 3)

4.3.1.7 JSP,s

Respecto a las páginas JSP, tal y como se ha comentado en el apartado anterior, también son susceptibles de ser accedidas directamente por los usuarios, pero en este caso la solución adoptada ha sido mucho más sencilla: ubicar éstas en carpetas no públicas.

La estructura definida de carpetas se muestra a continuación.



(Figura 4)

Si el lector inspecciona el contenido de las carpetas, podría sorprenderse ante la ubicación de algunos archivos, tales como las páginas de estilos CSS o los archivos externos con código JavaScript, así como ante la inexistencia de páginas con extensión HTML.

Los motivos aducidos para ubicarlos en esta zona son varios, el primero, ya comentado, debido ante todo a cuestiones de seguridad, el segundo, a la conveniencia de centralizar en una misma ubicación archivos relacionados. Esto ha sido posible gracias a la existencia de una directriz JSP muy práctica y cuya sintaxis es la siguiente:

<%@ include file="Filename" &>

No me extenderé sobre ésta, tan solo comentar que con esta directriz podemos acceder a todos los archivos dentro del WAR, incluyendo aquellos que no son públicamente visibles.

Como comentario final indicar que todas las páginas tratadas por el sistema son del tipo JSP, ya que, de otro modo, no habría sido posible la utilización de ciertas técnicas o, como mínimo, se habría añadido una mayor e innecesaria complejidad a la solución.

4.3.1.8 Limpieza y optimización de código

El documento de diseño facilita enormemente las tareas de implementación, pero considero utópico pensar que éste pueda dar solución a todas las posibles cuestiones que pudieran plantearse, ya que ello, en mi opinión, depende de muchos más factores que la simple voluntad de elaborar un buen documento, por ejemplo, las habilidades, experiencia y capacitación de los integrantes del equipo de desarrollo. Estas dificultades a la hora de abordar ciertas tareas pueden obligar a revisar el código tantas veces como fuera necesario, hasta obtener los resultados previstos, pero ¿es el resultado lo único importante?.

La experiencia acumulada en desarrollo me indica que no, ya que, aún cumplidos los requisitos establecidos, queda por determinar si se han utilizado los procedimientos más adecuados; no podemos olvidar que, tarde o temprano, nuestro desarrollo deberá dar solución a las nuevas necesidades del cliente y esta tarea siempre será mucho más sencilla si partimos de un código limpio y optimizado.

Una inspección visual puede determinar el grado de optimización y limpieza del código en aplicaciones sencillas, pero no en las más complejas con un alto número de clases y archivos externos.

Para las clases Java se puede automatizar esta tarea mediante la utilización de un aplicativo específico, como Jstyle, pero desconozco la existencia de software para realizar este tipo de comprobaciones con páginas JSP y a las que se han incorporado estilos CSS y funciones JavaScript, por lo tanto, en aplicativos que traten este tipo de elementos se hace conveniente imponer una serie de normas:

- ✓ La importación de código CSS y JavaScript desde archivos externos.
- ✓ La indicación en el documento de diseño del nombre y contenido de cada uno de estos, así como las páginas JSP que deberán hacer uso de ellos.

Estas sencillas reglas nos facilitarán la habitualmente árdua tarea de optimización y, a su vez, nos permitirán estar más cerca de obtener un producto final de calidad.

4.3.1.9 Control de transacciones

Se considera el control de transacciones un requisito obligatorio en cualquier desarrollo, con independencia de su complejidad o aunque haya sido explicitado en un documento, por lo tanto, creo innecesario realizar cualquier otro tipo de valoración al respecto, aunque sí comentar brevemente cómo ha sido resuelto.

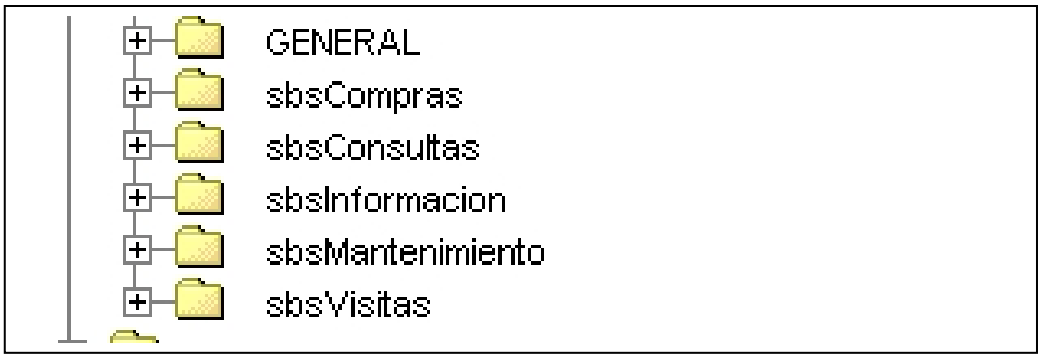
Cuando se obtiene una conexión se desactiva la confirmación implícita de transacciones, estando obligado el desarrollador a su confirmación o cancelación explícita, tanto en las modificaciones, el borrado, las inserciones y las lecturas. Respecto a éstas últimas, aunque teóricamente no sería necesario, en ciertas ocasiones y con otras bases de datos, como Informix, se han recibido excepciones de bloqueo en determinadas circunstancias.

4.3.1.10 División del proyecto en subsistemas

Siguiendo la célebre máxima "divide y vencerás", este aplicativo se descompone en subsistemas, todos ellos sin dependencias externas o, como mínimo, limitadas.

Esta división no sólo afecta a las diferentes páginas JSP y archivos externos relacionados, sino también a las clases Java del aplicativo. En concreto, el método seguido ha sido el que se muestra en la siguiente figura.

Comentario: las carpetas 'general' contienen los elementos comunes. Por ejemplo, en el caso de las páginas JSP, se incluyen aquellas páginas que son utilizadas por cualquier subsistema: página de error, etc.

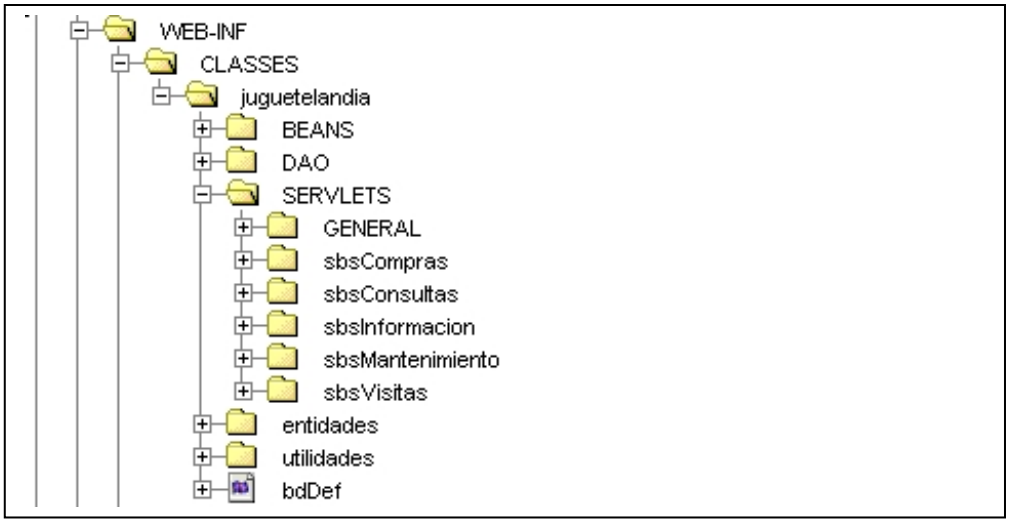


(Figura 5)

4.3.1.11 Paquetes Java

Las diferentes clases de las que se compone el aplicativo también han sido tratadas de forma similar a lo indicado para las JSP: división en subsistemas, aunque de una manera más sofisticada, ya que las clases de Java tienen más posibilidades de ser reutilizadas.

Aunque la aplicación se entrega sin crear físicamente los paquetes, ya que, entiendo, dificultaría un posible minucioso estudio del aplicativo; sin embargo la operación resulta del todo sencilla y no comporta dificultad alguna.



(Figura 6)

4.4 Capítulo 4 Líneas de trabajo futuras

Una de las características que caracteriza a los productos software es la posibilidad de realizar modificaciones en su comportamiento, sin limitaciones de ningún tipo, a excepción de aquellas que son inherentes a la propia tecnología.

Aunque sugerir o plantear cambios en un determinado proyecto pueda ser interpretado como algo negativo y poco profesional, en el mundo de la informática, concretamente en la ingeniería de programación, este aspecto supone precisamente todo lo contrario, ya que podría considerarse como un signo de innovación y superación. No podemos olvidar que el resultado de un proyecto de estas características es fruto, ya no tan sólo de la capacidad de sus autores, sino también de su brillantez a la hora de resolver ciertas cuestiones, algo que, en la mayoría de ocasiones, mejora a medida que aumenta la experiencia y se profundiza en la tecnología. No creo en las soluciones perfectas, ya que, si existieran, serían del todo innecesarias las modificaciones y la tecnología no evolucionaría al ritmo que lo hace.

La conclusión que se desprende es simple: los desarrollos de software evolucionan, o deberían hacerlo, en la medida que lo hace la tecnología, los conocimientos de sus autores y las necesidades de los usuarios.

Aunque esto es una realidad que, en mi opinión, pocos cuestionarían, aún existirían ciertas dudas que deberían ser planteadas al inicio de cualquier desarrollo, todas ellas con el único objetivo de alcanzar un grado de calidad adecuado, ya que, de lo contrario, no sería posible o, como mínimo, excesivamente dificultoso la evolución a la que se hace referencia.

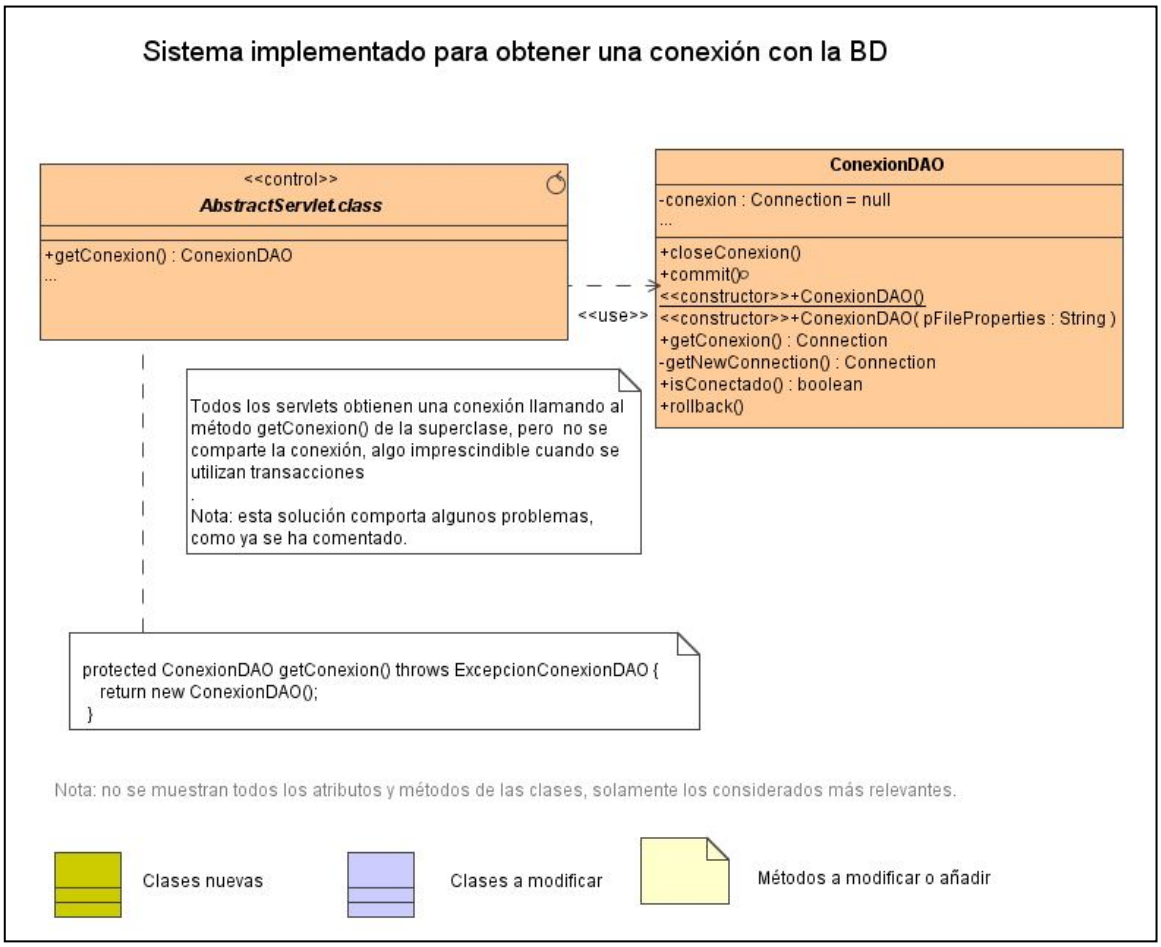
Los cambios que se plantean en este capítulo no entrañan excesiva dificultad, y facilitan de forma evidente el mantenimiento futuro del aplicativo y la incorporación de nuevas mejoras, resultado de las investigaciones y desarrollos de otros autores, aunque convendría en este punto remarcar que esto no sería posible sin el nivel de calidad exigido previamente durante todas las fases del proyecto.

4.4.1 Revisiones

En este apartado se recogen todas aquellas modificaciones que permitirán obtener un producto mucho más robusto, estable y, como no, extensible; todo ello sin un gran esfuerzo. Esto no sería posible o, como mínimo, muy dificultoso, si no se hubiera prestado previamente la debida atención a todo aquello que afecta o pudiera afectar a la extensibilidad del aplicativo.

4.4.2 Proveer de un 'Pool' de conexiones

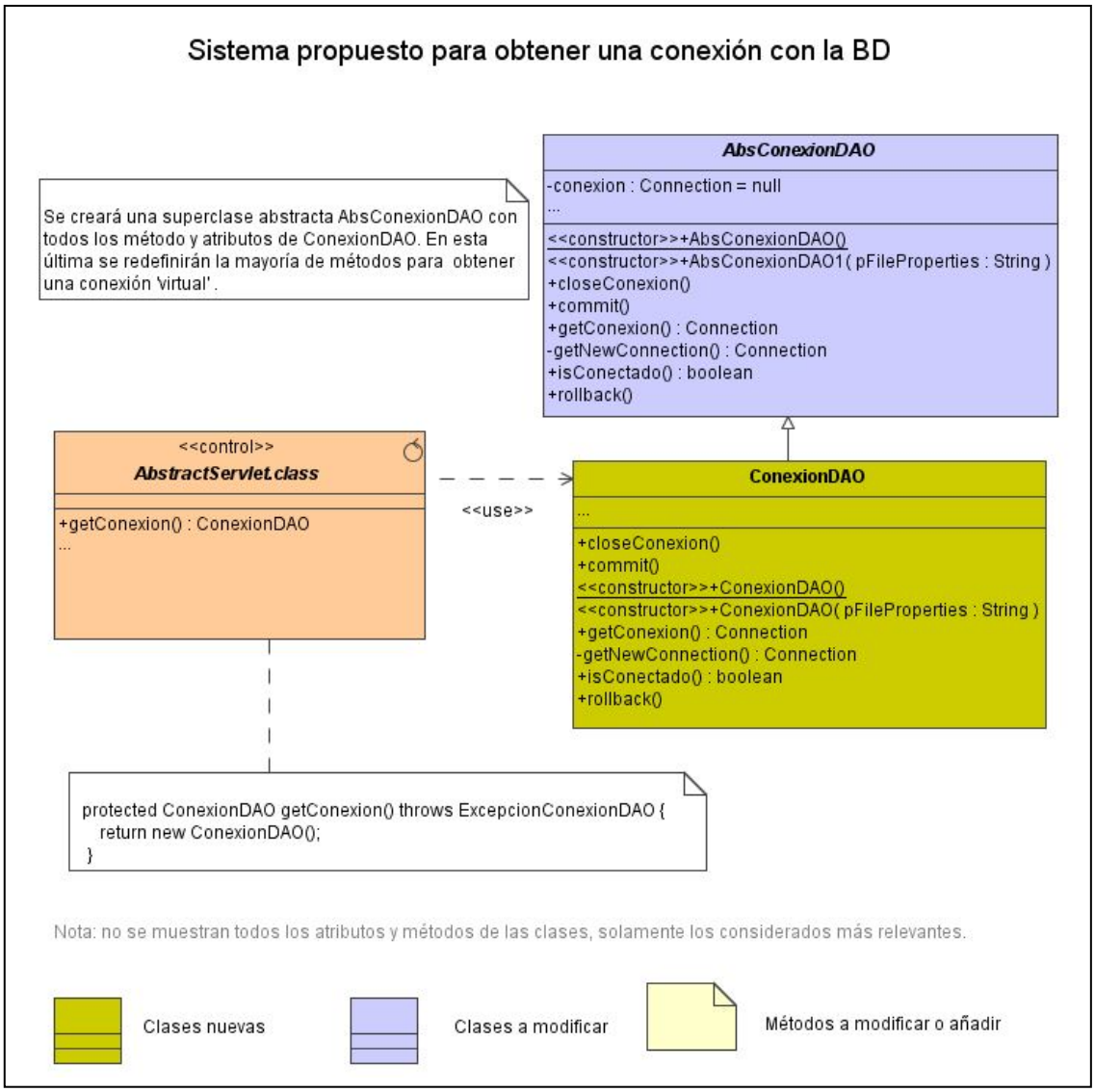
En la figura 7, en este mismo apartado, se describe y muestra cómo se ha tratado la obtención de las conexiones de la base de datos, pero este sistema comportaría serios problemas en un entorno productivo real.



(Figura 7)

Para concluir, la solución idónea a todos los niveles sería hacer reservas de conexiones, pero para ello se debe hacer uso del paquete javax.sql.ConnectionPoolDataSource, entre otros.

Aunque ésta implicaría realizar modificaciones en algunas clases, éstas tampoco entrañarían una excesiva dificultad.

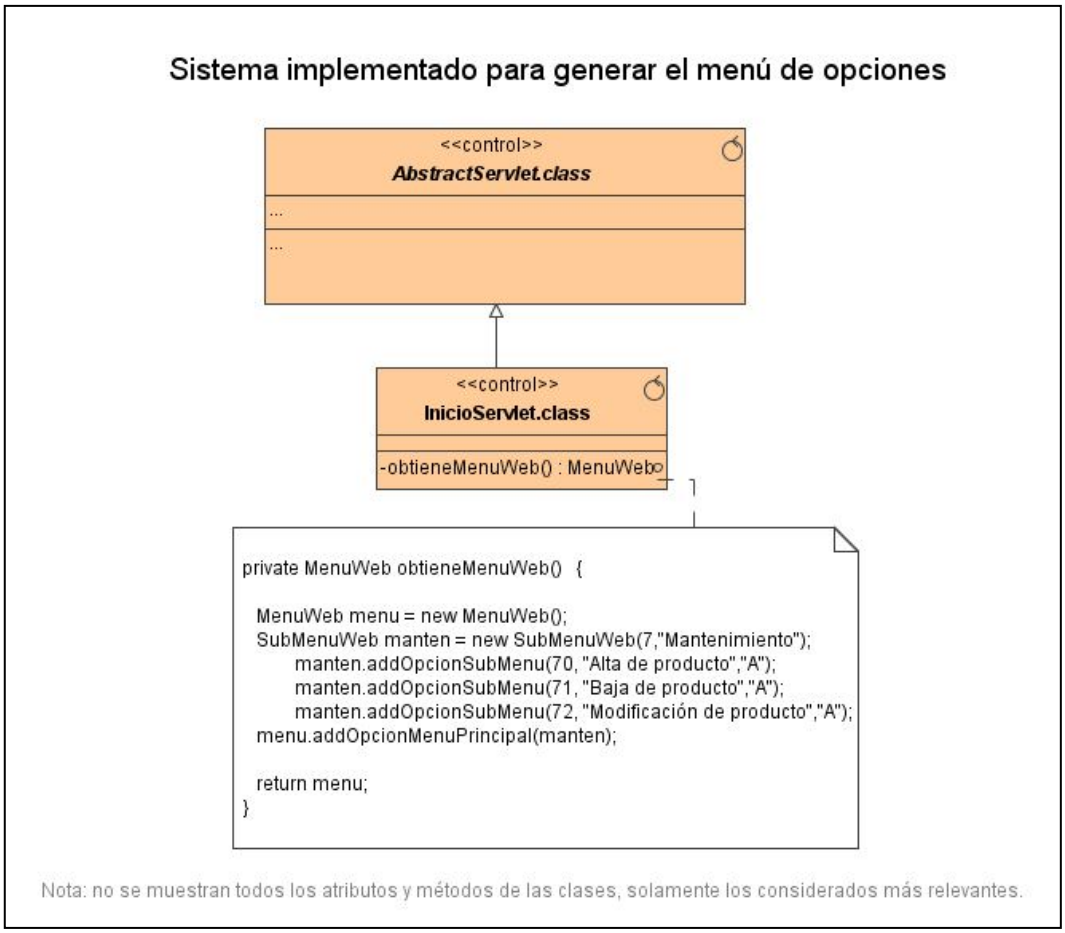


(Figura 8)

4.4.3 Carga del menú de opciones desde la base de datos

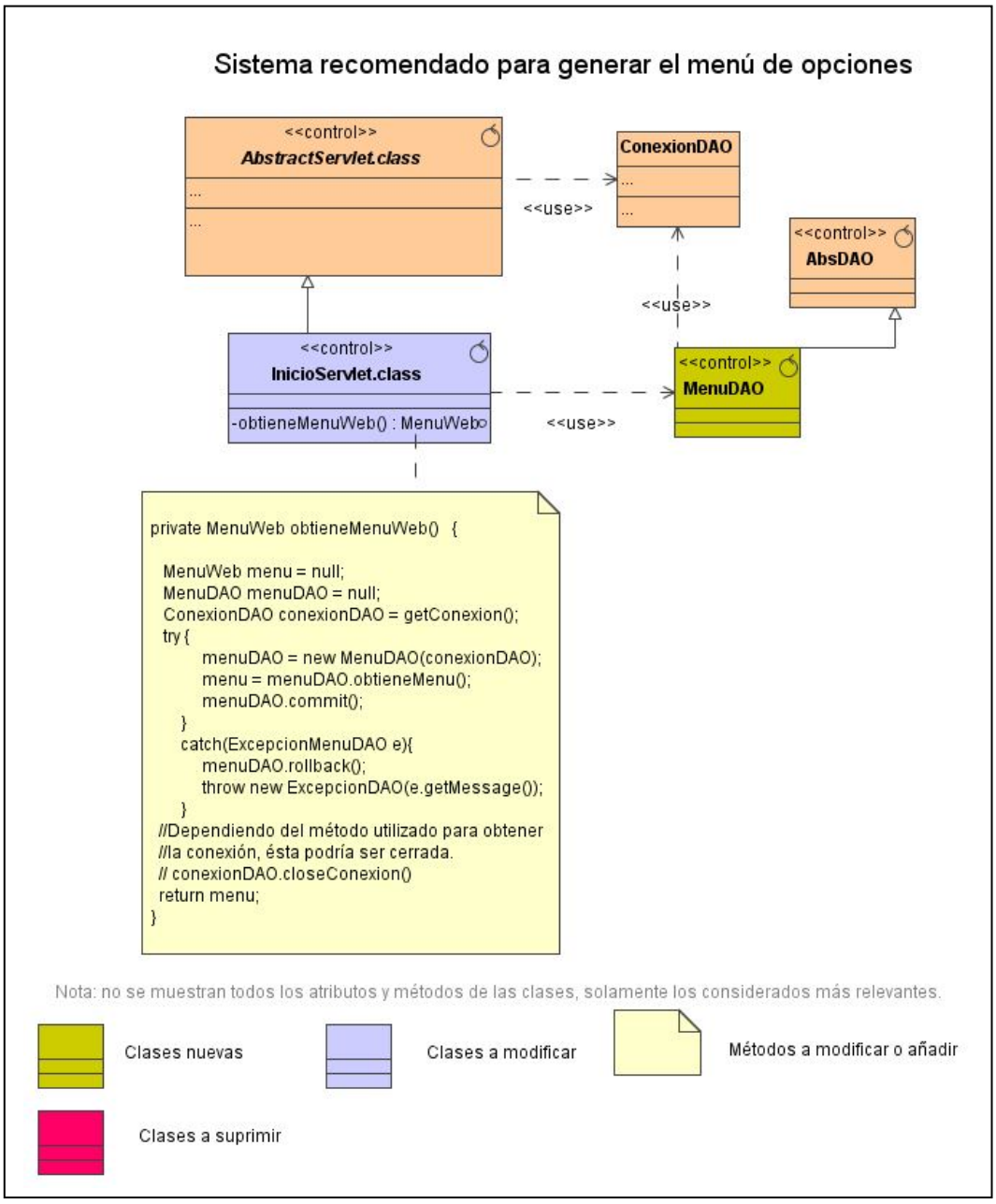
Esta resultaría una mejora interesante, ya que no sería necesario rectificar el servlet que genera el menú de opciones cada vez que se añada una nueva funcionalidad o se desee cambiar el perfil de acceso de una determinada opción.

El sistema implementado es el que se muestra en la figura.



(Figura 9)

Las modificaciones a realizar son las que se indican, consistente en modificar el método que obtiene el MenuWeb() del servlet 'InicioServlet' para que se obtenga el menú de la base de datos. Los cambios en este método ya se indican, así que solamente sería necesario crear la clase MenuDAO, algo muy sencillo si ya hemos definido las tablas en la BD.



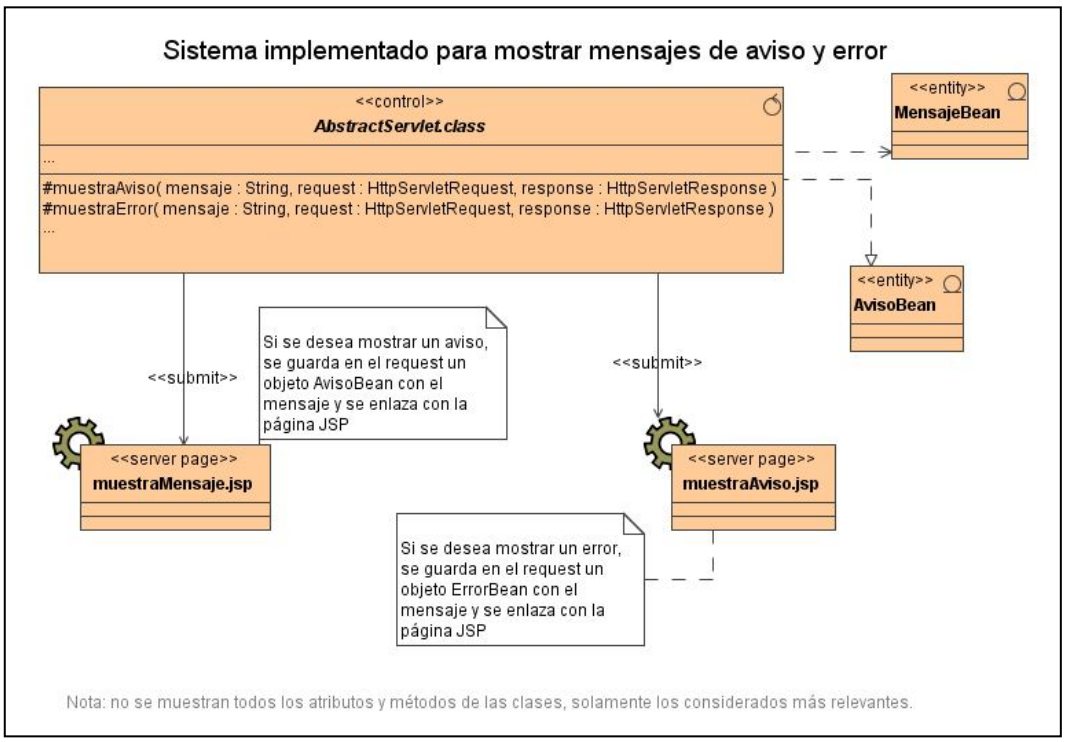
(Figura 10)

4.4.4 Simplificar las páginas JSP

4.4.4.1 Unificar Beans

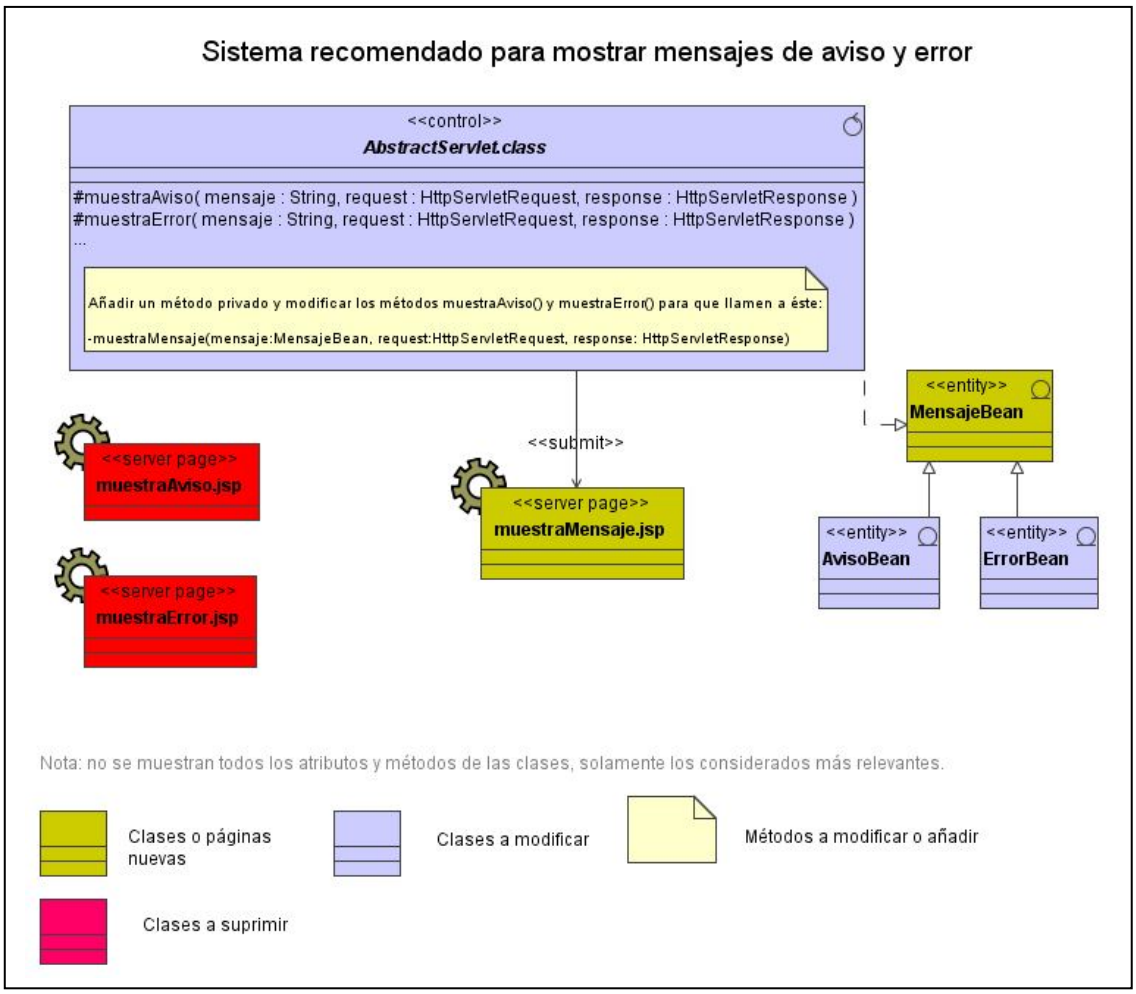
Observamos que el sistema implementado dispone de varios métodos para mostrar un mensaje, ya sea éste de aviso u error. Aunque ello no tiene mayores repercusiones en el conjunto del sistema, siempre es conveniente unificar ciertos elementos cuando no existe una buena razón para diferenciarlos.

Aunque la clase ErrorBean podría disponer de más métodos o realizar un tratamiento diferente de los mensajes, esto también podría obtenerse con una adecuada jerarquía de clases. No obstante, y en cualquier caso, con una página JSP sería suficiente.



(Figura 11)

La solución propuesta pasa por crear una jerarquía de Beans adecuada, añadir un método privado al servlet y unificar las páginas JSP.



(Figura 12)

4.4.4.2 Paquetes

Si el lector estudia con detenimiento la estructura creada llegará a la conclusión de que, aún sirviendo a los propósitos de este proyecto, ésta no es la idónea para la reutilización de algunos componentes generales, ya que éstas deberían estar situadas –dentro de un paquete- en el directorio definido por TOMCAT para archivos de este tipo, es decir, WEB-INF\lib.

Realizar lo anteriormente comentado tendrá sus implicaciones, ya que no resulta suficiente con crear un paquete y situarlo en el lugar indicado, puesto que es necesario rectificar el comando 'package' de todas las clases de la aplicación que utilizan alguna de las entidades que contiene.

4.4.5 Ampliación de funcionalidades

4.4.5.1 Mantenimiento de usuarios y otros

Las clases DAO incorporan todas ellas los métodos necesarios para realizar cualquier tipo de operación de mantenimiento sobre las tablas existentes, no solamente la de productos. Todas estas operaciones han sido verificadas mediante pruebas unitarias, por tanto se asegura su correcto funcionamiento.

En todo el aplicativo se utilizan las mismas páginas de estilos y scripts, así que solamente quedaría la creación de varias vistas y servlets, todo lo demás ya ha sido previamente implementado y verificado.

4.4.6 Poblar las listas de objetos con un limitado número de elementos.

En el aplicativo **eJuguetelandia** asumimos que el catálogo se compone de pocos productos, pero ¿qué sucedería si nuestro cliente dispusiera de un catálogo con varios cientos o miles de artículos?.

Los métodos de las clases DAO, tal y como han sido implementados, obtienen todos los productos de la base de datos, a excepción, claro está, de aquellos que limitan la selección, por lo tanto, si nos encontráramos ante esta situación, y antes siempre de llegar al extremo de modificar las clases, quizás sería interesante plantearse el impedir la selección de todos los productos de la BD. La solución propuesta pasaría por lo siguiente:

- ✓ Todo método que acceda a la BD para seleccionar una lista de elementos retornará, por ejemplo, una lista vacía si la selección excede de los parámetros prefijados, notificando esta circunstancia al usuario.

  La implementación de dicho aspecto sería muy sencilla si la BD dispone de mecanismos que permitan conocer, a priori, el número de elementos que se retornarán de una consulta.

4.5 Capítulo 5 Conclusiones

4.5.1 Herramientas de desarrollo

La experiencia me ha demostrado la importancia de seleccionar cuidadosamente las herramientas que serán utilizadas durante un proyecto de estas características, tanto en la fase de diseño como en la de implementación. Esto que, a priori, puede parecer insignificante, tendrá sus repercusiones en la duración de las tareas planificadas, llegando incluso a imposibilitar la entrega de los productos finales en los plazos establecidos. El motivo de ello está relacionado, en primer lugar, con las dificultades que comporta resolver ciertas anomalías de funcionamiento en un tiempo razonable, segundo, con la no adecuación de alguno de ellos a las normativas de referencia y que obligan a modificar manualmente ciertas partes del código, tercero, con las dificultades que comporta mantener, simultáneamente, el código del aplicativo y la documentación generada en anteriores fases.

En este proyecto se han utilizado diferentes tecnologías, algunas de ellas elevadas a recomendación, pero aún no extensamente utilizadas, como XHTML. Ésta ciertamente simplifica la creación de páginas, ya que permite su diseño usando un mismo lenguaje de marcas, aunque también comporta una serie de desventajas, como la obligatoriedad de realizar modificaciones manuales y la existencia de ciertas incompatibilidades con exploradores más antiguos, tales como MS IE5; muchas más si se utiliza MS IE4, NetScape o Motzilla. Aunque no ha sido posible evaluar un producto que permita construir ventanas XHTML -y aunque me consta que existen- desconozco la existencia de aplicativos que permitan construir y evaluar visualmente y de manera simultánea páginas JSP a las que se hayan añadido XHTML y JavaScript.

Esto último, expuesto como ejemplo práctico, permite hacerse una idea de las repercusiones que puede tener, a efectos de planificación, rectificar manualmente una serie de páginas JSP para adecuarlas a las necesidades del cliente.

4.5.2 Desarrollo del proyecto: ciclo de vida.

Aunque aplicar un ciclo de vida clásico a este proyecto ha permitido planificar las tareas de las que se compone y facilitar el seguimiento del mismo, no puede considerarse el más adecuado, ya que por idiosincrasia no tiene en cuenta las inevitables alteraciones que van sufriendo los distintos productos finales a medida que avanza el proyecto, ya sea por exigencias del cliente, por la detección de ciertas inconsistencias o, simplemente, porque la solución de diseño planteada originalmente no resultaba la más adecuada. En algunos de estos casos, las rectificaciones no comportan excesivos problemas, básicamente porque se centran en las ventanas de usuario y/o rectificaciones menores en ciertas entidades, pero en muchos otros, las modificaciones pueden llegar a afectar a la estructura general del aplicativo, algo que invitaría a replantear el desarrollo desde un principio; esto último del todo impensable en un proyecto de gran envergadura y complejidad.

Respecto al que nos ocupa, efectivamente nos hemos encontrado con todos los inconvenientes aquí expuestos y ello nos ha hecho reflexionar no solamente sobre la conveniencia de aplicar dicho ciclo de vida, sino también sobre la cualificación del diseñador y sobre su completo dominio de la tecnología a utilizar, no ya tan sólo a nivel teórico, sino también a nivel práctico, ya que de lo contrario no creo posible garantizar la calidad del producto final.

Existen otros ciclos de vida que estarían más en consonancia con las necesidades reales, como el de hacer exploraciones o el de construir prototipos, pero consideramos que éstos se centran en exceso en el cliente y en sus necesidades, descuidando o no incidiendo suficientemente en aspectos de futura relevancia, como la estructura interna del aplicativo y las posibilidades reales de reutilización y extensibilidad, algo fundamental si deseamos que nuestros desarrollos sean de calidad.

4.5.3 La calidad, garantía de extensibilidad y reutilización

En ‘Herramientas de desarrollo’, y a efectos meramente introductorios, se ha indicado la problemática de las páginas JSP a la hora de automatizar mediante un aplicativo la modificación de las diferentes ventanas, pero aunque ello fuera posible, nos encontraríamos con otra problemática: ¿cómo podemos diseñar de manera gráfica las ventanas de nuestro aplicativo, si nuestro deseo es concentrar los estilos aplicados en archivos externos?.

Lamentablemente no he encontrado una respuesta satisfactoria, y la única solución ha pasado por la creación manual de los estilos.

Esto último, aunque pueda pensarse que es innecesario y que dificulta el mantenimiento futuro, es sin lugar a dudas un signo de calidad, ya que permite realizar cierto tipo de rectificaciones prácticamente sin esfuerzo, aunque, evidentemente, éstas si así fuere deberían ser acometidas por un experto conocedor de la tecnología utilizada.

Respecto a la arquitectura J2EE, en concreto los servlets y las páginas JSP, y aunque no sería de aplicación todo lo comentado anteriormente, sí que podemos poner de relieve la importancia de crear, por ejemplo, una estructura de herencia acorde ya no sólo con nuestras necesidades actuales, sino también con aquellas que, previsiblemente, serán requeridas en un futuro. Es evidente que no tenemos la capacidad de prever todas ellas, pero sí las más evidentes. ¿Qué obtendremos si

aplicamos lo aquí indicado?. Primero, la satisfacción personal de haber realizado un buen desarrollo; segundo, profundizar y conocer las posibilidades de la tecnología existente; tercero, enfrentarnos a futuros desarrollos con seguridad; cuarto, disponer de la experiencia necesaria para elaborar diseños acertados, así como una planificación real de las diferentes tareas; quinto, poder reutilizar el modelo o parte del mismo en otros desarrollos; y en sexto lugar, simplificar las tareas de mantenimiento y la incorporación de nuevas mejoras. Todo esto, en resumen, es el resultado final de aquello a lo que siempre nos referimos como calidad.

La conclusión que se desprende de lo aquí comentado es la siguiente: alcanzar un alto nivel de calidad exige un gran esfuerzo personal y altamente motivado, y la asignación de gran parte de los recursos humanos y temporales a las fases de diseño e implementación, pero la recompensa posterior es grande: la garantía de extensibilidad y la reutilización del modelo.

4.5.4 Aplicativos para la web: ¿solución ideal?

Llegados a este punto, con la experiencia que me ha proporcionado este desarrollo y aún siendo consciente de carecer de conocimientos profundos en algunas de las tecnologías utilizadas, llegamos al convencimiento de que los aplicativos web no son la solución ideal para todo tipo desarrollos; ello se justifica a partir de lo siguiente:

- El desarrollo de un aplicativo web implica conocer diferentes tecnologías, algunas de ellas en evolución constante y que, como XHTML, no garantizan una compatibilidad al 100% con sus predecesoras, como el HTML.

- Existen diferencias considerables en la interpretación del lenguaje de marcas por los diversos navegadores existentes en el mercado, llegando incluso al no reconocimiento de algunas de ellas; también se produce una problemática similar con JavaScript.

- No es factible, sin utilizar tecnologías varias, diseñar una ventana con la riqueza visual proporcionada, por ejemplo, con la librería javax.swing.

- Es posible diseñar un aplicativo de "*n*" capas sin la obligatoriedad de utilizar páginas HTML o JSP, separando claramente la lógica de la presentación.

Los aplicativos web los consideraría imprescindibles para todos aquellos aplicativos en los que resulte imprescindible acceder desde cualquier punto de Internet, como JUGUETELANDIA, aunque, por los motivos ya comentados, para los mantenimientos de las diferentes tablas de la bases de datos -como la de productos- y otros aplicativos internos de la empresa cliente, quizás sería interesante plantearse otras tecnologías o, como mínimo, sopesar seriamente las ventajas e inconvenientes de cada una de ellas, ya que como es de todos sabido, no existe la solución ideal.

5 Glosario

• **Autenticarse**	Validar ante el sistema la identidad del usuario
• **Catálogo**	Conjunto de productos u artículos ofertados
• **Cliente ligero**	Programas que no consumen muchos recursos de los equipos informáticos locales y que permiten manejar con comodidad un determinado aplicativo
• **Cuenta de usuario**	Nombre o identificador del usuario para autenticarse ante el sistema
• **Edad recomendada**	Edad recomendada por el fabricante para utilizar su producto
• **EJB**	Enterprise Java Beans
• **Funcionalidad**	Opciones que proporciona un programa
• **J2EE**	Java 2 Enterprise Edition
• **JSP**	Java Server Pages
• **MVC**	Modelo, vista, controlador.
• **Open Source**	De fuente libre.
• **Páginas dinámicas**	Lo contrario de las páginas estáticas
• **Páginas estáticas**	Páginas, normalmente web, que contienen información estática, es decir, no variable y no obtenida de otras fuentes.
• **Perfil**	Características o atribuciones específicas que diferencian a un usuario de otro.
• **Servidor**	Equipo central dedicado a atender las peticiones de los puestos de trabajo
• **Thin Client**	Cliente ligero
• **Usuario registrado**	Usuario que ha cumplimentado la ficha de registro
• **Vistas**	Páginas o interficies que se le muestran al usuario

6 Bibliografía

1. Jacob Nielsen. Prentice may (2002). Usabilidad. Diseño de sitios Web.

2. Elizabeth Castro. Anaya Multimedia (2003) .HTML con XHTML y CSS.

3. Lázaro Issi Camy. Anaya Multimedia. JAVASCRIPT

4. Anaya Multimedia (2002). Profesional Programación Java Server con J2EE Ed. 1.3.

5. Universidad de Cádiz. Manual de HTML
   http://www.uca.es/manual-html/font.htm

6. Roberto Canales Mora. JSPs en Tomcat y MVC
   http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=jsps

7. Primeros pasos con Tomcat
   http://bulma.net/body.phtml?nIdNoticia=1888

8. Tomcat faqs
   http://jakarta.apache.org/tomcat/faq/misc.html

9. MySQL Manual. SQL Statement Syntax
   http://dev.mysql.com/doc/mysql/en/SQL_Syntax.html

10. MySQL Control Center
    http://www.desarrolloweb.com/articulos/898.php?manual=34

11. Singletons en Java, el patrón instancia única
    http://www.elrincondelprogramador.com/default.asp?pag=articulos%2Fleer.asp

12. The W3C Markup Validation Service
    http://validator.w3.org/

7 Anexos

7.1 Vistas de usuario

Recientemente y dentro de los estudios de EITG en esta misma universidad, tuve la oportunidad y el placer de participar en el estudio y confección de una serie de páginas Web estáticas y cuya finalidad no era otra que la de facilitar a personas con deficiencias visuales la navegación por las diferentes páginas del aplicativo.

Los resultados obtenidos fueron excelentes y la experiencia muy grata, entre otros motivos, porque me hizo ser consciente de las dificultades que en este mundo de prisas padecen ciertas personas, así como también disponer de los conocimientos necesarios para poder opinar sobre la idoneidad de un determinado diseño.

Se adjunta un breve resumen de aquellos aspectos de diseño que considero más importantes para facilitar la navegación y un uso mucho más intuitivo del sistema, todos ellos ampliados en el documento Análisis de Requisitos.

7.1.1 Consideraciones generales

- No hacer uso de los marcos laterales, especialmente si estos son mostrados de manera permanente en la parte izquierda de la ventana.
- Toda información accesoria, pero que se requiere con frecuencia, tales como los enlaces a otros contenidos, por norma general estará situadas en la parte superior de las diferentes ventanas y se resaltará mediante el uso del subrayado y el color.
- La información que se pueda requerir desde un determinado lugar estará escondida, pero accesible mediante una determinada acción del usuario.
- Las combinaciones de colores de fondo y primer plano tendrán el suficiente contraste para que puedan ser vistas por cualquier tipo de usuario.
- Marcar convenientemente los enlaces consecutivos mediante el uso del carácter '|'.

7.1.2 Menú de opciones

- Los grupos de opciones se situarán longitudinalmente en la parte superior de la ventana.
- Todas las opciones del menú principal y submenús estarán siempre visibles.
- Con objeto de facilitar la orientación, las diferentes opciones de los menús se situarán, de manera fija, en lugares específicos de la pantalla, diferenciando el menú de los submenús mediante el uso de color y posición.

7.1.3 Ventana de contenidos

- El color de fondo será un degradado del azul, casi blanco; para el primer plano se utilizará el negro o un color suficientemente oscuro. Con esto se conseguirá un contraste óptimo.

- La cabecera de la ventana siempre tendrá la misma estructura, evitando las barras de desplazamiento.

- El menú principal estará siempre fijo y en el mismo lugar. Los submenús se situarán siempre en lugar bien visible y evitando, en la medida de lo posible, el despliegue de los mismos.