
UNIVERSIDADE DA CORUÑA

FACULTADE DE INFORMÁTICA

*Departamento de Tecnoloxías da Información e as Comunicacións*

PROYECTO FIN DE CARRERA DE INGENIERÍA INFORMÁTICA


# ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras.

Autor: Carlos Sánchez González

Director: Fernando Bellas Permuy

A Coruña, Septiembre de 2004

| | |
|---|---|
| Título | ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras. |
| Clase | Proyecto clásico de ingeniería |
| Autor | Carlos Sánchez González |
| Director | Fernando Bellas Permuy |
| Tribunal | |
| Fecha de lectura | |
| Calificación | |

# Dedicatoria

A mi familia.

# Tabla de contenidos

# Lista de figuras

# Lista de tablas

# Lista de ejemplos

# Resumen

En los últimos años las aplicaciones web están ganando adeptos frente a las aplicaciones tradicionales *standalone*. La facilidad de uso de cara al usuario final, la facilidad de administración centralizada y el auge del comercio electrónico vía internet son algunas de las causas. Esto ha hecho que hayan surgido gran número de soluciones para su desarrollo.

El presente proyecto plantea el desarrollo de un sistema de gestión de una empresa dedicada a la venta al mayor dentro del comercio textil, utilizando para ello las últimas y más innovadoras soluciones desarrolladas dentro de la comunidad *open source*, minimizando el tiempo de desarrollo a la vez que maximizando la calidad del producto.

Para el desarrollo del sistema se contará con el apoyo de una empresa del sector que facilitará los datos y la experiencia necesarios para realizar un sistema ajustado al mundo real con el objetivo de implantarlo en la propia empresa.

El resultado de este proyecto es a su vez devuelto a la comunidad como software *open source* certificado por la OSI (*Open Source Initiative*) bajo Apache License Version 2.0. La vía escogida ha sido la creación de un proyecto con toda la información en el sito web de Sourceforge http://oness.sourceforge.net, comunidad dedicada a toda clase de proyectos *open source.*

**Palabras clave**: ONess, J2EE, web, Maven, Spring, Hibernate, Struts, AOP, programación orientada a aspectos

# Parte I. Introducción

# Tabla de contenidos

# Capítulo 1. Escenario

Los sistemas de gestión son un concepto que existe desde los principios de la informática, pero el coste de un sistema que se ajuste a las necesidades particulares relega su uso a medianas o grandes empresas capaces de soportarlo. Las pequeñas empresas son mayoría en España, y su uso de las tecnologías de la información es realmente escaso, teniendo que subsistir con simples programas genéricos, tecnológicamente muy limitados, para cualquier tipo de negocio con la consiguiente pérdida de productividad debida a que en lugar de adaptar un sistema a sus procesos acaban realizando lo contrario. En el caso de la empresa textil, que juega un gran papel en la economía gallega, esto es si cabe peor debido a las especiales características de la mercancía con la que negocian, muy diferenciada de otros tipos de productos.

Un sistema de gestión debe permitir gestionar la información de la empresa desde el inicio de los procesos hasta su finalización. En el ámbito en el que se engloba ONess, una empresa mayorista, los principales procesos son:

- ventas

  En el sector textil las ventas se realizan principalmente mediante visita de un representante a los clientes (tiendas) en su lugar de trabajo. El presentar unas muestras físicas de los distintos modelos se hace prácticamente imprescindible dadas las características de los productos.

  Estos pedidos deben ser considerados cuanto antes dado que retrasos en disponer de esta información afectará negativamente a la toma de decisiones posteriores (por ejemplo mercancía que se acaba y se sigue vendiendo)

- compras

  El proceso es similar al de ventas pero a la inversa. Dado que en el sector textil el grueso de las compras se realiza con una anticipación superior a los seis meses tiene gran importancia el conocer los pedidos que se encuentran en curso.

- servicio

  Tanto los productos comprados como los vendidos pueden ser servidos en distintas ocasiones según distintas condiciones.

- gestión de stock

  Los productos textiles suelen ser complicados de gestionar. La inmensa mayoría no disponen de códigos de barras y se referencian habitualmente mediante su código de modelo, talla y color.

Otros aspectos que deberían estar presentes en cualquier sistema y que realmente son difíciles de encontrar en sistemas no hechos a medida son:

- seguridad

  Se debe disponer de una política de seguridad para permitir en todo momento que los usuarios sólo tengan acceso a los recursos a los que están permitidos.

- auditoría

  Todos los cambios que se realizan en el sistema deben estar registrados, junto con el usuario que los ha realizado y el momento en el que lo han hecho.

- accesibilidad

  Directamente relacionada con la seguridad, un sistema de gestión debe ser accesible tanto para los usuarios locales situados en una intranet como desde cualquier otra localización vía, por ejemplo, internet, y potencialmente utilizando cualquier tipo de dispositivo, sean ordenadores personales, PDAs o incluso móviles.

**ONess** cubre todos estos aspectos, y, aunque su inicial aplicación es la gestión en empresas del sector textil, su modularidad hace que partes del sistema sean aplicables a muy distintos ámbitos de las tecnologías de la información.

Algunas de sus características son:

- Es fácilmente extensible, el tiempo de creación de nueva funcionalidad es realmente muy pequeño.
- Está basado en software *open source* ampliamente utilizado y probado, con una gran comunidad de usuarios, lo que proporciona un soporte de gran calidad.
- Dispone de mecanismos de seguridad y gestión de permisos, lo que permite mostrar distinta información según el rol del usuario así como definir las acciones que pueden realizar y las que no.
- Promueve la separación de roles entre los desarrolladores gracias al patrón MVC (Model-View-Controller) y la separación en capas y módulos funcionales.
- Soporta la práctica totalidad de los sistemas de gestión de bases de datos relacionales: MySQL, PostgreSQL, Oracle,...
- Es internacionalizable, los mensajes de la aplicación pueden ser traducidos a cualquier idioma y cada usuario los verá automáticamente en el suyo.

- Es *open source*, proporcionando entre otras ventajas una gran flexibilidad.

## 1.1. Objetivos

El presente proyecto aborda la creación del sistema de gestión para el sector textil, proporcionando la funcionalidad básica para su uso, a la vez que se proporcionan unas bases sólidas para el desarrollo de cualquier tipo de aplicación Java que desee aprovechar las oportunidades que brindan las últimas tecnologías *open source*.

Se desarrollarán los módulos necesarios para proporcionar las siguientes funcionalidades:

- gestión de contactos, que englobará tanto a clientes y proveedores, gestionando la información necesaria para realizar las operaciones comerciales.

- gestión de inventario, con la funcionalidad necesaria para trabajar con modelos y productos.

- gestión de pedidos, cubriendo el proceso desde la recepción/realización de un pedido, pasando por su envío, hasta su facturación final.

# Capítulo 2. ONess: una visión global

## 2.1. Visión global

ONess proporciona una arquitectura estable que integra las soluciones más innovadoras del mundo *open source* en un proceso de desarrollo ágil.

Está compuesto de una serie de módulos, cada uno de ellos con una funcionalidad específica.

- *common* es la base del resto de los sistemas
- *user* gestión de usuarios
- *party* agrupa la funcionalidad referida a los contactos, tanto clientes como proveedores, empleados,...
- *inventory* contiene toda la información referida a los productos y su gestión: modelos, productos, almacenes, precios,...
- *order* abarca el proceso desde pedido hasta factura pasando por albarán.

Cada uno de estos módulos se divide a su vez arquitecturalmente en *model* y *webapp*, conteniendo el primero la capa modelo y el segundo las capas controlador y vista de la arquitectura MVC que se discutirá posteriormente en el Capítulo 3, *Justificación*.

En cuanto a funcionalidad, el núcleo del sistema proporciona características como control de acceso a usuarios, con auditoría completa de los cambios realizados por éstos, o soporte para el desarrollo de vistas personalizadas según el tipo de navegador, lo que permite que se adapte fácilmente la totalidad del sistema a dispositivos móviles como PDAs.

Cada uno de los módulos proporciona una funcionalidad bien definida, cubriendo los procesos habituales de una empresa mayorista, como son la gestión de clientes y proveedores, el control de existencias y los procesos de compras y ventas de mercancías. Todo ello desarrollado sobre el núcleo del sistema y tomando ventaja de las características que éste proporciona de forma transparente, obteniendo así un proceso de desarrollo de nueva funcionalidad realmente rápido.

## 2.2. Base tecnológica

ONess se basa principalmente en los siguientes estándares y tecnologías que se describirán en la Sección 6.3, “Tecnologías”.

- J2EE (Java 2, Enterprise Edition), versión empresarial de la plataforma de desarrollo de aplicaciones Java 2, de Sun Microsystems, que aporta estándares tecnológicos para la creación de aplicaciones web (servlets, JSP), el acceso a bases de datos (JDBC), el tratamiento de XML (JAXP), servicios de directorio (JNDI), etc.
- Maven, gestor de información de proyecto.
- AspectJ, extensión del lenguaje Java proporcionando características de orientación a aspectos.
- Spring Framework, entorno para el desarrollo de aplicaciones fomentando el patrón inversión de control y la integración entre tecnologías.
- Acegi Security System for Spring, proyecto que incorpora características de seguridad dentro de Spring.
- Hibernate, un mapeador objeto-relacional que proporciona un puente entre la programación orientada a objetos y los sistemas de gestión de bases de datos relacionales.
- XDoclet, generador de código a partir de atributos en el código fuente.
- Struts, framework que proporciona la capa controlador y parte de la capa vista en aplicaciones web basadas en la arquitectura MVC.
- JSP (Java Server Pages) para el desarrollo del interfaz.
- Tiles utilizado para el desarrollo de páginas web en componentes.
- CSS (Cascading Style Sheets), hojas de estilo para desarrollar interfaces de usuario más potentes y separar de mejor manera la presentación de los datos.
- JSTL (JavaServer Pages Standard Tag Library), librería de etiquetas estándar que sustituye el código Java por el uso de etiquetas XML en las páginas JSP.
- JUnit, DBUnit, JMock y StrutsTestCase para realizar los tests del sistema.
- Sistemas de Gestión de Bases de Datos, pudiendo trabajar actualmente con tres importantes sistemas como son Oracle, PostgreSQL y MySQL.
- Tomcat, contenedor de aplicaciones web que proporciona la implementación estándar del API servlets y JSP, aunque cualquier otra implementación puede ser usada.

# Parte II. Justificación y metodología

# Tabla de contenidos

# Capítulo 3. Justificación

## 3.1. Aplicaciones web

En los últimos años las aplicaciones web han sufrido un gran auge gracias en gran parte a Internet y la proliferación de sitios web, sobre todo con el fin de fomentar el comercio electrónico.

Su facilidad de administración centralizada las hace ideales tanto para su despliegue en internet como en intranets corporativas.

La facilidad de uso de los interfaces web y el hecho de que cada día más personas están acostumbradas a la navegación por internet hace que el tiempo de aprendizaje se reduzca considerablemente respecto a aplicaciones tradicionales *standalone*.

El auge de multitud de soluciones o *frameworks* open source hace que su desarrollo sea sencillo y que un gran número de desarrolladores tengan experiencia con ellos. Otro hecho a tener en cuenta es que una vez realizada una aplicación web para uso interno de una empresa, por ejemplo en una intranet, el poner esa funcionalidad, o incluso funcionalidades nuevas, a disposición de empleados o el público general tiene un coste mínimo a la vez que una potencial proyección mundial.

## 3.2. Aplicaciones en capas

La estrategia tradicional de utilizar aplicaciones compactas causa gran cantidad de problemas de integración en sistemas software complejos como pueden ser los sistemas de gestión de una empresa o los sistemas de información integrados consistentes en más de una aplicación. Estas aplicaciones suelen encontrarse con importantes problemas de escalabilidad, disponibilidad, seguridad, integración...

Para solventar estos problemas se ha generalizado la división de las aplicaciones en capas que normalmente serán tres: una capa que servirá para guardar los datos (base de datos), una capa para centralizar la lógica de negocio (modelo) y por último una interfaz gráfica que facilite al usuario el uso del sistema.

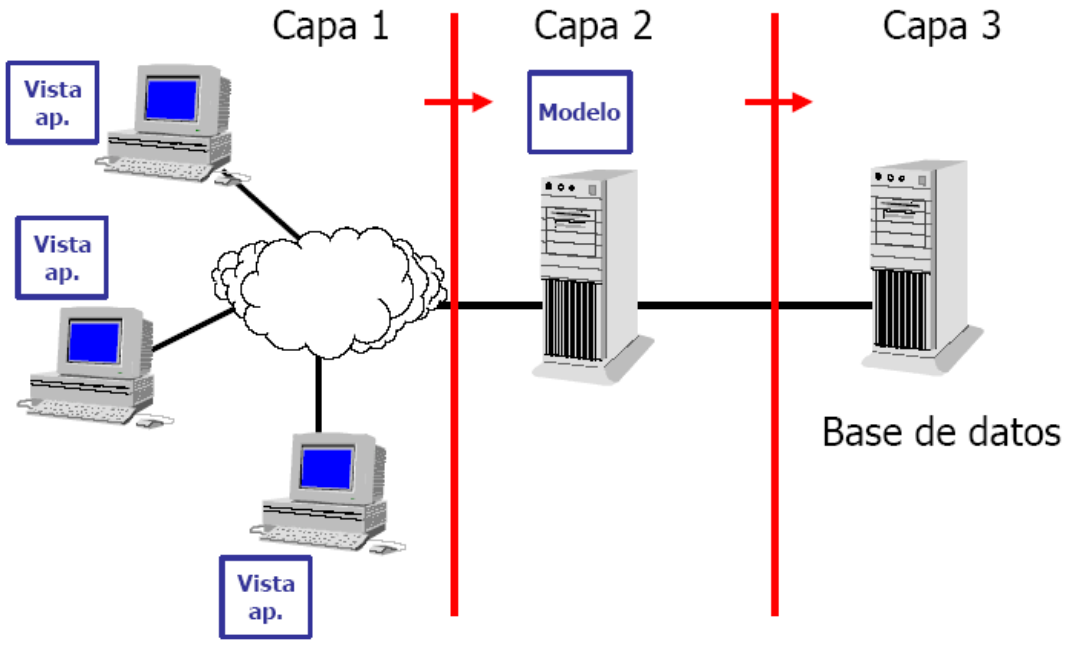


**Figura 3.1. Arquitecura en tres capas**

Si establecemos una separación entre la capa de interfaz gráfica (cliente), replicada en cada uno de los entornos de usuario, y la capa modelo, que quedaría centralizada en un servidor de aplicaciones, según el diagrama que podemos ver en la Figura 3.1, “Arquitecura en tres capas”, obtenemos una potente arquitectura que nos otorga algunas ventajas:

- Centralización de los aspectos de seguridad y transaccionalidad, que serían responsabilidad del modelo.
- No replicación de lógica de negocio en los clientes: esto permite que las modificaciones y mejoras sean automáticamente aprovechadas por el conjunto de los usuarios, reduciendo los costes de mantenimiento.
- Mayor sencillez de los clientes.

Si intentamos aplicar esto a las aplicaciones web, debido a la obligatoria sencillez del software cliente que será un navegador web, nos encontramos con una doble posibilidad:

- Crear un modelo de 4 capas, tal y como puede verse en la Figura 3.2, “Arquitectura web en cuatro capas”, separando cliente, servidor web, modelo y almacén de datos. Esto nos permite una mayor extensibilidad en caso de que existan también clientes no web en el sistema, que trabajarían directamente contra el servidor del modelo.

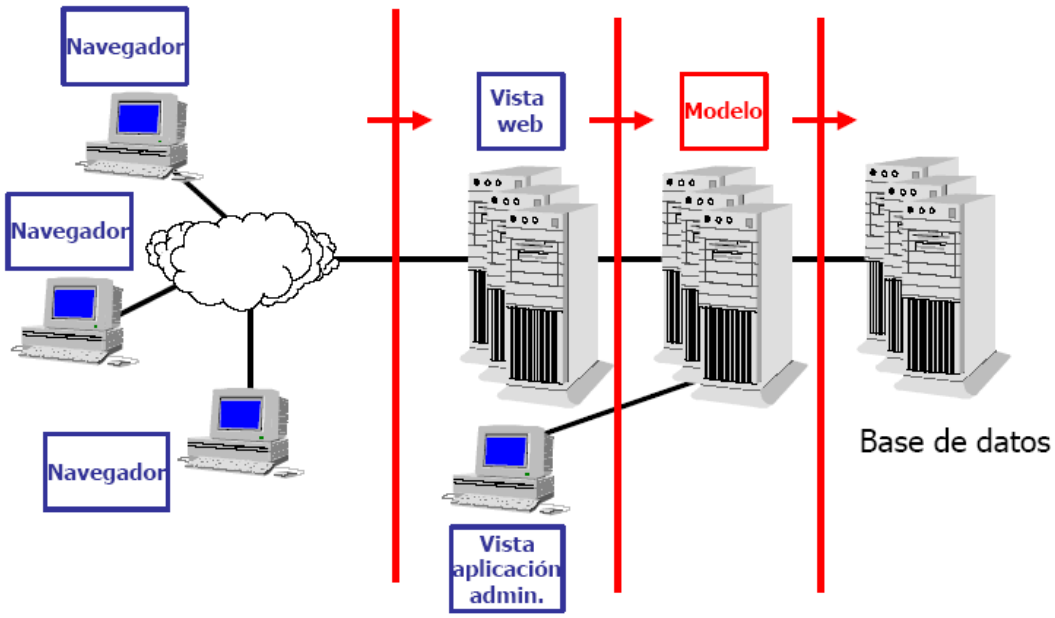


**Figura 3.2. Arquitectura web en cuatro capas**

- Mantener el número de capas en 3, como se ve en la Figura 3.3, “Arquitectura web en 3 capas”, integrando interfaz web y modelo en un mismo servidor aunque conservando su independencia funcional. Ésta es la distribución en capas más común en las aplicaciones web.

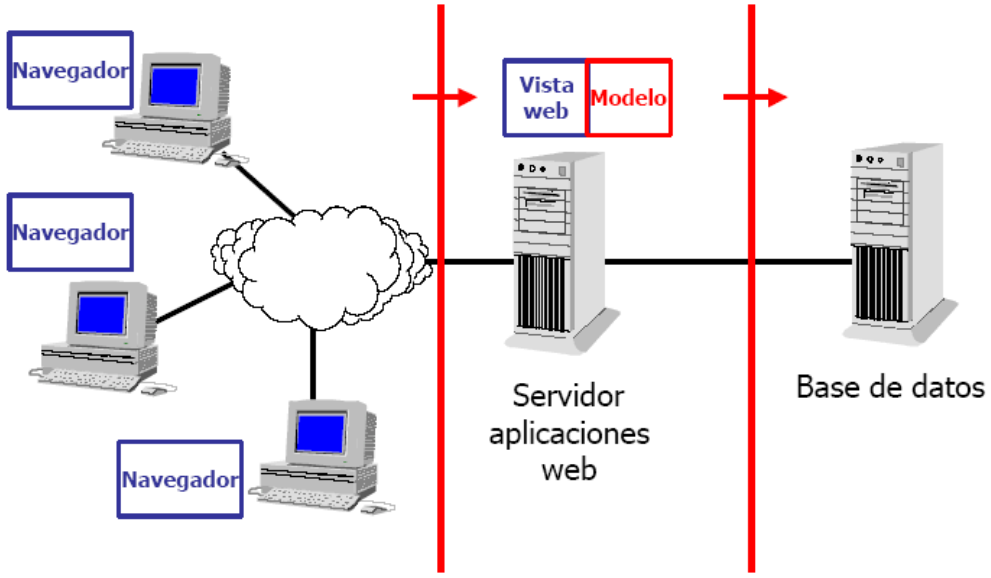


**Figura 3.3. Arquitectura web en 3 capas**

La arquitectura utilizada por el proyecto ONess define tres capas bien diferenciadas, si bien gracias al soporte que proporciona en framework Spring permite la implantación de una capa de modelo basada en Enterprise JavaBeans (EJB) que posibilitase una arquitectura de cuatro capas y unas funcionalidades más orientadas a grandes sistemas o sistemas críticos como pueden ser replicación, trabajo en cluster,...

### 3.2.1. La arquitectura MVC y Model 2

La arquitectura Model-View-Controller surgió como patrón arquitectónico para el desarrollo de interfaces gráficos de usuario en entornos Smalltalk. Su concepto se basaba en separar el modelo de datos de la aplicación de su representación de cara al usuario y de la interacción de éste con la aplicación, mediante la división de la aplicación en tres partes fundamentales:

- El modelo, que contiene la lógica de negocio de la aplicación.
- La vista, que muestra al usuario la información que éste necesita.
- El controlador, que recibe e interpreta la interacción del usuario, actuando sobre modelo y vista de manera adecuada para provocar cambios de estado en la representación interna de los datos, así como en su visualización.

Esta arquitectura ha demostrado ser muy apropiada para las aplicaciones web y especialmente adaptarse bien a las tecnologías proporcionadas por la plataforma J2EE, de manera que:

- El modelo, conteniendo lógica de negocio, sería modelado por un conjunto de clases Java, existiendo dos claras alternativas de implementación, utilizando objetos java tradicionales llamados POJOs (*Plain Old Java Objects*) o bien utilizando EJB (*Enterprise JavaBeans*) en sistemas con unas mayores necesidades de concurrencia o distribución.
- La vista proporcionará una serie de páginas web dinámicamente al cliente, siendo para él simples páginas HTML. Existen múltiples frameworks que generan estas páginas web a partir de distintos formatos, siendo el más extendido el de páginas JSP (*JavaServer Pages*), que mediante un conjunto de tags XML proporcionan un interfaz sencillo y adecuado a clases Java y objetos proporcionados por el servidor de aplicaciones. Esto permite que sean sencillas de desarrollar por personas con conocimientos de HTML. Entre estos tags tienen mención especial la librería estándar JSTL (*JavaServer Pages Standard Tag Library*) que proporciona una gran funcionalidad y versatilidad.
- El controlador en la plataforma J2EE se desarrolla mediante servlets, que hacen de intermediarios entre la vista y el modelo, más versátiles que los JSP para esta función al estar escritos como clases Java normales, evitando mezclar código visual (HTML, XML...)

con código Java. Para facilitar la implementación de estos servlets también existe una serie de frameworks que proporcionan soporte a los desarrolladores, entre los que cabe destacar Struts, que con una amplia comunidad de usuarios se ha convertido en el estándar *de facto* en este rol.

Con todo lo anterior, el funcionamiento de una aplicación web J2EE que utilice el patrón arquitectural MVC se puede descomponer en una serie de pasos:

1. El usuario realiza una acción en su navegador, que llega al servidor mediante una petición HTTP y es recibida por un servlet (controlador). Esa petición es interpretada y se transforma en la ejecución de código java que delegará al modelo la ejecución de una acción de éste.

2. El modelo recibe las peticiones del controlador, a través de un interfaz o fachada que encapsulará y ocultará la complejidad del modelo al controlador. El resultado de esa petición será devuelto al controlador.

3. El controlador recibe del modelo el resultado, y en función de éste, selecciona la vista que será mostrada al usuario, y le proporcionará los datos recibidos del modelo y otros datos necesarios para su transformación a HTML. Una vez hecho esto el control pasa a la vista para la realización de esa transformación.

4. En la vista se realiza la transformación tras recibir los datos del controlador, elaborando la respuesta HTML adecuada para que el usuario la visualice.

Esta arquitectura de aplicaciones otorga varias ventajas clave al desarrollo de aplicaciones web, destacando:

- Al separar de manera clara la lógica de negocio (modelo) de la vista permite la reusabilidad del modelo, de modo que la misma implementación de la lógica de negocio que maneja una aplicación pueda ser usado en otras aplicaciones, sean éstas web o no.

- Permite una sencilla división de roles, dejando que sean diseñadores gráficos sin conocimientos de programación o desarrollo de aplicaciones los que se encarguen de la realización de la capa vista, sin necesidad de mezclar código Java entre el código visual que desarrollen (tan sólo utilizando algunos tags, no muy diferentes de los usados en el código HTML).

#### 3.2.1.1. Model 2

Sun Microsystems, creadora de la plataforma Java, acuñó el término Model 2 para referirse al modelo arquitectural recomendado para las aplicaciones web desarrolladas sobre J2EE.

Dicha arquitectura consiste en el desarrollo de una aplicación según el patrón Model-View-Controller, pero especificando que el controlador debe estar formado por un único servlet, que centralice el control de todas las peticiones al sistema, y que basándose en la URL de la petición HTTP y en el estado actual del sistema, derive la gestión y control de la petición a una determinada acción de entre las registradas en la capa controlador. Esta centralización del controlador en un único punto de acceso se conoce como patrón front controller.

Las ventajas que este patrón ofrece provienen de la capacidad de gestionar en un único punto la aplicación de filtros a las peticiones, las comprobaciones de seguridad, la realización de logs, etc.

## 3.3. Open Source

La decisión de liberar el código fuente de este proyecto tiene principalmente como objetivo el obtener una proyección internacional y los contactos para mantener una colaboración con personas destacadas en la comunidad Java.

Por ser *open source*, la decisión de utilizar esta aplicación ha de tener en cuenta ciertos hechos relevantes remarcados en [Wheeler04]:

- Cuota de mercado
- Fiabilidad
- Rendimiento
- Escalabilidad
- Seguridad
- Costo total de la propiedad (el costo de la compra cuando se le incluye todos los gastos que lo acompañan como por ejemplo: mantenimiento, soporte, capacitación de usuario y etc, especialmente en ordenadores)
- Reducción de riesgos y desventajas respecto a soluciones propietarias.
- Evita conflictos de licencias y costes de administración.
- Flexibilidad
- Apoyo a la innovación

Temores infundados que se asocian al software libre:

- Las soluciones propietarias están mejor soportadas que las soluciones abiertas.
- El software propietario da a los usuarios más derechos legales que el software libre.
- Los programas open source son meros plagios de las soluciones comerciales.
- El software libre expone al usuario a un mayor riesgo de abandono.
- El software libre no es viable económicamente.
- El software libre acabará con la industria del software.
- El software libre es incompatible con el capitalismo.
- Si en una categoría sólo existe software libre eso acabaría con la competencia.
- El software libre destruye la propiedad intelectual.
- No hay realmente mucho software libre.
- La posibilidad de ver o cambiar el código fuente realmente no es importante para mucha gente.
- Open source es tan sólo una campaña anti-Microsoft.
- No hay nada gratis, seguro que el software libre tiene trampa.

# Capítulo 4. Estudio: otras aplicaciones de gestión *open source*

En este capítulo se intentará dar una visión de dos proyectos *open source* estudiados antes de haber emprendido la creación del proyecto ONess, tanto desde el punto de vista de funcionalidad como de tecnologías empleadas. Como ya se ha comentado anteriormente es difícil encontrar un sistema con un soporte adecuado al comercio textil, y estos dos proyectos no son una excepción. Si no fuera por ello el proyecto *Open for Business* sería realmente una alternativa muy adecuada a un sistema propietario.

## 4.1. Compiere

El proyecto Compiere [http://www.compiere.org] ha desarrollado un sistema de gestión, llevado a cabo por la empresa ComPiere, Inc.

Su modelo de negocio se basa en la liberación del código mientras se cobran los servicios de soporte que ofrecen mediante una red de asociados.

### 4.1.1. Características

Sus principales características son:

- Amplia Funcionalidad.
- Número uno en descargas.
- Soporte no gratuito.
- Complejidad enorme, tanto como de cara al usuario como a los desarrolladores.
- Proporciona clientes web como standalone.
- No ofrece un soporte específico para el comercio textil.

### 4.1.2. Tecnología empleada

En cuanto a tecnología la principal característica que llama la atención es que requiere el gestor de bases de datos Oracle 9i2, lo que prácticamente anula todos los beneficios que proporciona el hecho de ser *open source*.

- Requiere el sistema gestor de base de datos Oracle.
- Utiliza procedimientos almacenados en la base de datos, con lo que su migración a otros sistemas es prácticamente inviable.
- Utiliza EJB sobre el servidor JBoss.

## 4.2. Open for Business OFBiz

El proyecto Open for Business [http://www.ofbiz.org] es desarrollado principalmente por David E. Jones y Andy Zeneski.

### 4.2.1. Características

Sus principales características son:

- Una gran funcionalidad que abarca multitud de ámbitos de negocio.
- Incluye facilidades para comercio electrónico.
- No es independiente del sistema gestor de base de datos, pero soporta las principales *open source*: MaxDB, PostgreSQL e Hypersonic SQL.
- Es un proyecto maduro comenzado a mediados de 2001.
- Interfaz de usuario no demasiado complicado con un diseño uniforme.
- No ofrece un soporte específico para el comercio textil.

### 4.2.2. Tecnología empleada

Utiliza las mismas tecnologías que en sus comienzos, desarrollando muchas partes del sistema específicamente para el proyecto lo que causa que no lleguen a ser conocidas ni usadas por la comunidad de desarrolladores, provocando una barrera de entrada de cara a su adopción y adaptación.

- Utiliza gran cantidad de proyectos open source.
- Usa un amplio conjunto de estándares.
- Utiliza un motor de persistencia realizado específicamente para el proyecto basado en metadatos en xml, aunque en sus planes está migrar a Hibernate en próximas versiones.

- Utiliza gran cantidad de metadatos, lo que aunque aumenta su flexibilidad hace que sea realmente complejo.
- El núcleo central del proyecto no ha variado desde sus inicios, no ha aprovechado el gran número de soluciones que han surgido desde entonces.

# Capítulo 5. Metodología

Para la realización del presente trabajo se han seguido las directrices marcadas por la metodología XP (*Extreme Programming*).

## 5.1. Una introducción a *eXtreme Programming*

Esta metodología promueve los siguientes valores:

- Comunicación

  El eXtreme Programming se nutre del ancho de banda más grande que se puede obtener cuando existe algún tipo de comunicación: la comunicación directa entre personas. Es muy importante entender cuales son las ventajas de este medio. Cuando dos (o más) personas se comunican directamente pueden no solo consumir las palabras formuladas por la otra persona, sino que también aprecian los gestos, miradas, etc. que hace su compañero. Sin embargo, en una conversación mediante el correo electrónico, hay muchos factores que hacen de esta una comunicación, por así decirlo, mucho menos efectiva.

- Coraje

  El coraje es un valor muy importante dentro de la programación extrema. Un miembro de un equipo de desarrollo extremo debe de tener el coraje de exponer sus dudas, miedos, experiencias sin "embellecer" éstas de ninguna de las maneras. Esto es muy importante ya que un equipo de desarrollo extremo se basa en la confianza para con sus miembros. Faltar a esta confianza es una falta más que grave.

- Simplicidad

  Dado que no se puede predecir como va a ser en el futuro, el software que se esta desarrollando; un equipo de programación extrema intenta mantener el software lo más sencillo posible. Esto quiere decir que no se va a invertir ningún esfuerzo en hacer un desarrollo que en un futuro pueda llegar a tener valor. En el XP frases como "...en un futuro vamos a necesitar..." o "Haz un sistema genérico de..." no tienen ningún sentido ya que no aportan ningún valor en el momento.

- Feedback

  La agilidad se define (entre otras cosas) por la capacidad de respuesta ante los cambios que se van haciendo necesarios a lo largo del camino. Por este motivo uno de los valore que nos hace más ágiles es el continuo seguimiento o feedback que recibimos a la hora de desarrollar

en un entorno ágil de desarrollo. Este feedback se toma del cliente, de los miembros del equipo, en cuestión de todo el entorno en el que se mueve un equipo de desarrollo ágil.

Para ello se fundamenta en las siguientes doce prácticas [Jeffries01] [CanosLetelierPenades03] [Ferrer03]:

1. Planificación incremental

   La Programación Extrema asume que la planificación nunca será perfecta, y que variará en función de cómo varíen las necesidades del negocio. Por tanto, el valor real reside en obtener rápidamente un plan inicial, y contar con mecanismos de feedback que permitan conocer con precisión dónde estamos. Como es lógico, la planificación es iterativa: un representante del negocio decide al comienzo de cada iteración qué características concretas se van a implementar.

   El objetivo de la XP es generar versiones de la aplicación tan pequeñas como sea posible, pero que proporcionen un valor adicional claro, desde el punto de vista del negocio. A estas versiones se las denomina releases.

   Una release cuenta con un cierto número de historias. La historia es la unidad de funcionalidad en un proyecto XP, y corresponde a la mínima funcionalidad posible que tiene valor desde el punto de vista del negocio. Durante cada iteración se cierran varias historias, lo que hace que toda iteración añada un valor tangible para el cliente.

   Es fundamental en toda esta planificación la presencia de un representante del cliente, que forma parte del equipo y que decide cuáles son las historias más valiosas. Estas historias son las que se desarrollarán en la iteración actual.

   La obtención de feedback que permita llevar a cabo estimaciones precisas es fundamental. Se hacen estimaciones para cada historia, de modo que en cuanto se comienzan a tener datos históricos, éstos se utilizan para hacer que las siguientes estimaciones sean más precisas.

   Como se puede ver, y como siempre ocurre con la Programación Extrema, el enfoque utilizado para llevar a cabo la planificación es eminentemente pragmático. Gran parte de la eficacia de este modelo de planificación deriva de una división clara de responsabilidades, que tiene en cuenta las necesidades del negocio en todo momento. Dentro de esta división, el representante del cliente tiene las siguientes responsabilidades:

   - Decidir qué se implementa en cada release o iteración.
   - Fijar las fechas de fin de la release, recortando unas características o añadiendo otras.

- Priorizar el orden de implementación, en función del valor de negocio.

Las responsabilidades del equipo de desarrollo son las siguientes:

- Estimar cuánto tiempo llevará una historia: este feedback es fundamental para el cliente, y puede llevarle a reconsiderar qué historias se deben incluir en una iteración.
- Proporcionar información sobre el coste de utilizar distintas opciones tecnológicas.
- Organizar el equipo.
- Estimar el riesgo de cada historia.
- Decidir el orden de desarrollo de historias dentro de la iteración.

2. Testing

   La ejecución automatizada de tests es un elemento clave de la XP. Existen tanto tests internos (o tests de unidad ), para garantizar que el mismo es correcto, como tests de aceptación, para garantizar que el código hace lo que debe hacer. El cliente es el responsable de definir los tests de aceptación, no necesariamente de implementarlos. Él es la persona mejor cualificada para decidir cuál es la funcionalidad más valiosa.

   El hecho de que los tests sean automatizados es el único modo de garantizar que todo funciona: desde el punto de vista de la XP, si no hay tests, las cosas sólo funcionan en apariencia. Aún más, si un test no está automatizado, no se le puede considerar como tal.

   El objetivo de los tests no es corregir errores, sino prevenirlos. Por ejemplo, los tests siempre se escriben antes que el código a testear, no después: esto aporta un gran valor adicional, pues fuerza a los desarrolladores a pensar cómo se va a usar el código que escriben, poniéndolos en la posición de consumidores del software. Elaborar los tests exige pensar por adelantado cuáles son los problemas más graves que se pueden presentar, y cuáles son los puntos dudosos. Esto evita muchos problemas y dudas, en lugar de dejar que aparezcan "sobre la marcha".

   Un efecto lateral importante de los tests es que dan una gran seguridad a los desarrolladores: es posible llegar a hacer cambios más o menos importantes sin miedo a problemas inesperados, dado que proporcionan una red de seguridad. La existencia de tests hace el código muy maleable.

3. Programación en parejas

   La XP incluye, como una de sus prácticas estándar, la programación en parejas. Nadie programa en solitario, siempre hay dos personas delante del ordenador. Ésta es una de las

características que más se cuestiona al comienzo de la adopción de la XP dentro de un equipo, pero en la práctica se acepta rápidamente y de forma entusiasta.

El principal argumento contra la programación en parejas es que es improductiva. Esta idea se basa en el hecho de que dos programadores "programan el doble por separado". Esto sería así si no fuera por las siguientes razones:

- El hecho de que todas las decisiones las tomen al menos dos personas proporciona un mecanismo de seguridad enormemente valioso.
- Con dos personas responsabilizándose del código en cada momento, es menos probable que se caiga en la tentación de dejar de escribir tests, etc., algo fundamental para mantener el código en buena forma. Es muy difícil que dos personas se salten tareas por descuido o negligencia.
- El hecho de programar en parejas permite la dispersión de know-how por todo el equipo. Este efecto es difícil de conseguir de otro modo, y hace que la incorporación de nuevos miembros al equipo sea mucho más rápida y eficaz.
- El código siempre está siendo revisado por otra persona. La revisión de código es el método más eficaz de conseguir código de calidad, algo corroborado por numerosos estudios, muchos de los cuáles son anteriores a la Programación Extrema.
- En contra de lo que pueda parecer, los dos desarrolladores no hacen lo mismo: mientras el que tiene el teclado adopta un rol más táctico, el otro adopta un rol más estratégico, preguntándose constantemente si lo que se está haciendo tiene sentido desde un punto de vista global.
- Los datos indican que la programación en parejas es realmente más eficiente. Si bien se sacrifica un poco de velocidad al comienzo, luego se obtiene una velocidad de crucero muy superior. Esto contrasta con lo que ocurre en la mayor parte de los proyectos, en los que se arranca con una velocidad enorme pero rápidamente se llega a un estado muy parecido a la parálisis, en el que progresos cada vez más pequeños consumen cantidades de tiempo cada vez más grandes. Todos conocemos proyectos que se pasan el 50% del tiempo en el estado de "finalizado al 90%".

4. Refactorización

A la hora de la verdad, el código de la mayor parte de las aplicaciones empieza en un razonable buen estado, para luego deteriorarse de forma progresiva. El coste desorbitado del mantenimiento, modificación y ampliación de aplicaciones ya existente se debe en gran parte a este hecho.

Uno de los objetivos de la XP es mantener la curva de costes tan plana como sea posible, por lo que existen una serie de mecanismos destinados a mantener el código en buen estado, modificándolo activamente para que conserve claridad y sencillez. A este proceso básico para mantener el código en buena forma se le llama refactorización.

La refactorización no sólo sirve para mantener el código legible y sencillo: también se utiliza cuando resulta conveniente modificar código existente para hacer más fácil implementar nueva funcionalidad.

5. Diseño simple

Otra práctica fundamental de la Programación Extrema es utilizar diseños tan simples como sea posible. El principio es "utilizar el diseño más sencillo que consiga que todo funcione". Se evita diseñar características extra porque a la hora de la verdad la experiencia indica que raramente se puede anticipar qué necesidades se convertirán en reales y cuáles no. La XP nos pide que no vivamos bajo la ilusión de que un diseño puede resolver todas o gran parte de las situaciones futuras: lo que parece necesario cambia con frecuencia, es difícil acertar a priori.

Es obvio que, si no vamos a anticipar futuras necesidades, debemos poder modificar el diseño si alguna de estas se materializa. La XP soporta estas modificaciones gracias a los tests automatizados. Estos permiten hacer cambios importantes gracias a la red de protección que proporcionan. La refactorización, que hace que el código existente sea claro y sencillo, también ayuda a hacer factibles las modificaciones.

La XP define un "diseño tan simple como sea posible" como aquél que:

- Pasa todos los tests.
- No contiene código duplicado.
- Deja clara la intención de los programadores (enfatiza el qué, no el cómo) en cada línea de código.
- Contiene el menor número posible de clases y métodos.

6. Propiedad colectiva del código

La XP aboga por la propiedad colectiva del código. En otras palabras, todo el mundo tiene autoridad para hacer cambios a cualquier código, y es responsable de ellos. Esto permite no tener que estar esperando a otros cuando todo lo que hace falta es algún pequeño cambio.

Por supuesto, cada cuál es responsable de las modificaciones que haga. El principio básico es "tú lo rompes, tú lo arreglas, no importa si está en el código propio o en el de otros".

Por último, vale la pena tener en cuenta que la existencia de tests automatizados impide que se produzca un desarrollo anárquico, al ser cada persona responsable de que todos los tests se ejecuten con éxito al incorporar los cambios que ha introducido al programa.

7. Integración continua

   En muchos casos la integración de código produce efectos laterales imprevistos, y en ocasiones la integración puede llegar a ser realmente traumática, cuando dejan de funcionar cosas por motivos desconocidos. La Programación Extrema hace que la integración sea permanente, con lo que todos los problemas se manifiestan de forma inmediata, en lugar de durante una fase de integración más o menos remota.

   La existencia de una fase de integración separada tiene dos efectos laterales indeseables: se empieza a hacer codificación "yo-yo", en la que todo el mundo modifica código "sólo para que funcione, ya lo ajustaremos", y hace que se acumulen defectos. Evitar que se acumulen defectos es muy importante para la XP, como lo es el conseguir que los defectos que cada programador inyecta los elimine él mismo.

8. Cliente en el equipo

   Algunos de los problemas más graves en el desarrollo son los que se originan cuando el equipo de desarrollo toma decisiones de negocio críticas. Esto no debería ocurrir, pero a la hora de la verdad con frecuencia no se obtiene feedback del cliente con la fluidez necesaria: el resultado es que se ha de optar por detener el avance de los proyectos, o por que desarrollo tome una decisión de negocio. Por otra parte, los representantes del negocio también suelen encontrarse con problemas inesperados debido a que tampoco reciben el feedback adecuado por parte de los desarrolladores.

   La XP intenta resolver este tipo de problemas integrando un representante del negocio dentro del equipo de desarrollo. Ésta persona siempre está disponible para resolver dudas y para decidir qué y qué no se hace en cada momento, en función de los intereses del negocio. Debido a su inmersión dentro del equipo, y a que es él quien decide qué y qué no se hace, junto con los tests que verifican si la funcionalidad es la correcta y deseada, esta persona obtiene un feedback absolutamente realista del estado del proyecto.

9. Releases pequeñas

   Siguiendo la política de la XP de dar el máximo valor posible en cada momento, se intenta liberar nuevas versiones de las aplicaciones con frecuencia. Éstas deben ser tan pequeñas como sea posible, aunque deben añadir suficiente valor como para que resulten valiosas para el cliente.

10. Semanas de 40 horas

La Programación Extrema lleva a un modo de trabajo en que el equipo está siempre al 100%. Una semana de 40 horas en las que se dedica la mayor parte del tiempo a tareas que suponen un avance puede dar mucho de sí, y hace innecesario recurrir a sobreesfuerzos -excepto en casos extremos.

Además, el sobreesfuerzo continuado pronto lleva a un rendimiento menor y a un deterioro de la moral de todo el equipo.

11. Estándares de codificación

    Para conseguir que el código se encuentre en buen estado y que cualquier persona del equipo pueda modificar cualquier parte del código es imprescindible que el estilo de codificación sea consistente. Un estándar de codificación es necesario para soportar otras prácticas de la XP.

    Sin embargo, la XP también es pragmática en esto, y apuesta por establecer un número mínimo de reglas: el resto se irán pactando de-facto. Esto evita un ejercicio inicial más o menos estéril.

12. Uso de Metáforas

    La comunicación fluida es uno de los valores más importantes de la Programación Extrema: la programación en parejas, el hecho de incorporar al equipo una persona que represente los intereses del negocio y otras prácticas son valiosas entre otras cosas porque potencian enormemente la comunicación.

    Para conseguir que la comunicación sea fluida es imprescindible, entre otras cosas, utilizar el vocabulario del negocio. También es fundamental huir de definiciones abstractas. Dicho de otro modo, la XP no pretende seguir la letra de la ley, sino su espíritu. Dentro de este enfoque es fundamental buscar continuamente metáforas que comuniquen intenciones y resulten descriptivas, enfatizando el qué por delante del cómo.

La metodología XP es una metodología *ágil* [AgileManifesto]:

- Los individuos e interacciones son más importantes que los procesos y herramientas.

  Al individuo y las interacciones del equipo de desarrollo sobre el proceso y las herramientas. La gente es el principal factor de éxito de un proyecto software. Es más importante construir un buen equipo que construir el entorno. Muchas veces se comete el error de construir primero el entorno y esperar que el equipo se adapte automáticamente. Es mejor crear el equipo y que éste configure su propio entorno de desarrollo en base a sus necesidades.

- Software que funcione es más importante que documentación exhaustiva.

Desarrollar software que funciona más que conseguir una buena documentación. La regla a seguir es “no producir documentos a menos que sean necesarios de forma inmediata para tomar un decisión importante”. Estos documentos deben ser cortos y centrarse en lo fundamental.

- La colaboración con el cliente es más importante que la negociación de contratos.

  La colaboración con el cliente más que la negociación de un contrato. Se propone que exista una interacción constante entre el cliente y el equipo de desarrollo. Esta colaboración entre ambos será la que marque la marcha del proyecto y asegure su éxito.

- La respuesta ante el cambio es más importante que el seguimiento de un plan.

  Responder a los cambios más que seguir estrictamente un plan. La habilidad de responder a los cambios que puedan surgir a los largo del proyecto (cambios en los requisitos, en la tecnología, en el equipo, etc.) determina también el éxito o fracaso del mismo. Por lo tanto, la planificación no debe ser estricta sino flexible y abierta.

## 5.2. Ciclo de vida de un proyecto XP

Las fases que define *eXtreme Programming* pueden verse en la Figura 5.1, “Fases de un proyecto en eXtreme Programming”.

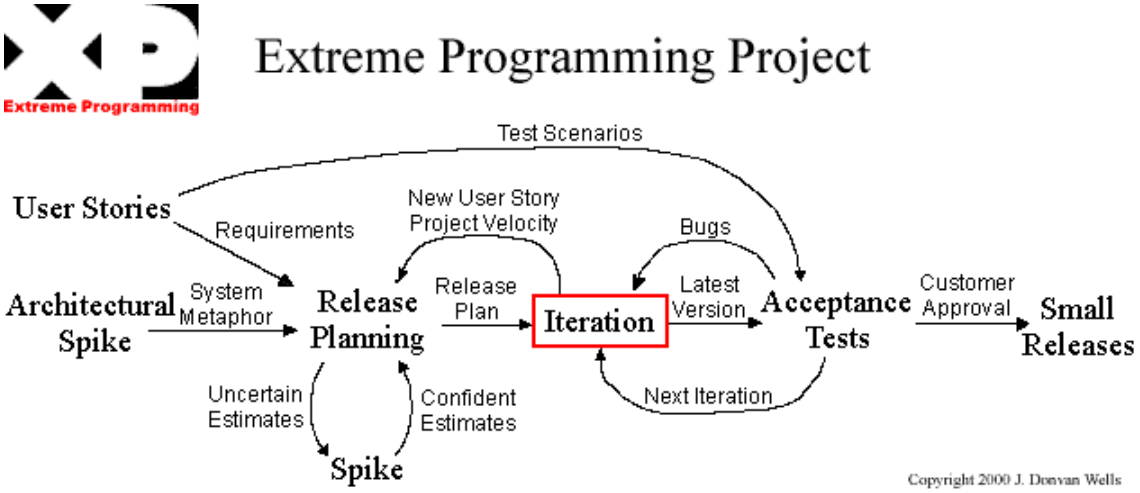


**Figura 5.1. Fases de un proyecto en eXtreme Programming**

El ciclo de vida ideal de XP consiste de seis fases [LetelierPenades03]:

I. Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

II. Planificación de la Entrega (*Release*)

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, el equipo de desarrollo mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración. La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

III. Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la

iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero llevadas a cabo por parejas de programadores.

IV. Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase. Es posible que se rebaje el tiempo que toma cada iteración, de tres a una semana. Las ideas que han sido propuestas y las sugerencias son documentadas para su posterior implementación (por ejemplo, durante la fase de mantenimiento).

V. Mantenimiento

Mientras la primera versión se encuentra en producción, el proyecto XP debe mantener el sistema en funcionamiento al mismo tiempo que desarrolla nuevas iteraciones. Para realizar esto se requiere de tareas de soporte para el cliente. De esta forma, la velocidad de desarrollo puede bajar después de la puesta del sistema en producción. La fase de mantenimiento puede requerir nuevo personal dentro del equipo y cambios en su estructura.

VI. Muerte del Proyecto

Es cuando el cliente no tiene más historias para ser incluidas en el sistema. Esto requiere que se satisfagan las necesidades del cliente en otros aspectos como rendimiento y confiabilidad del sistema. Se genera la documentación final del sistema y no se realizan más cambios en la arquitectura. La muerte del proyecto también ocurre cuando el sistema no genera los beneficios esperados por el cliente o cuando no hay presupuesto para mantenerlo.

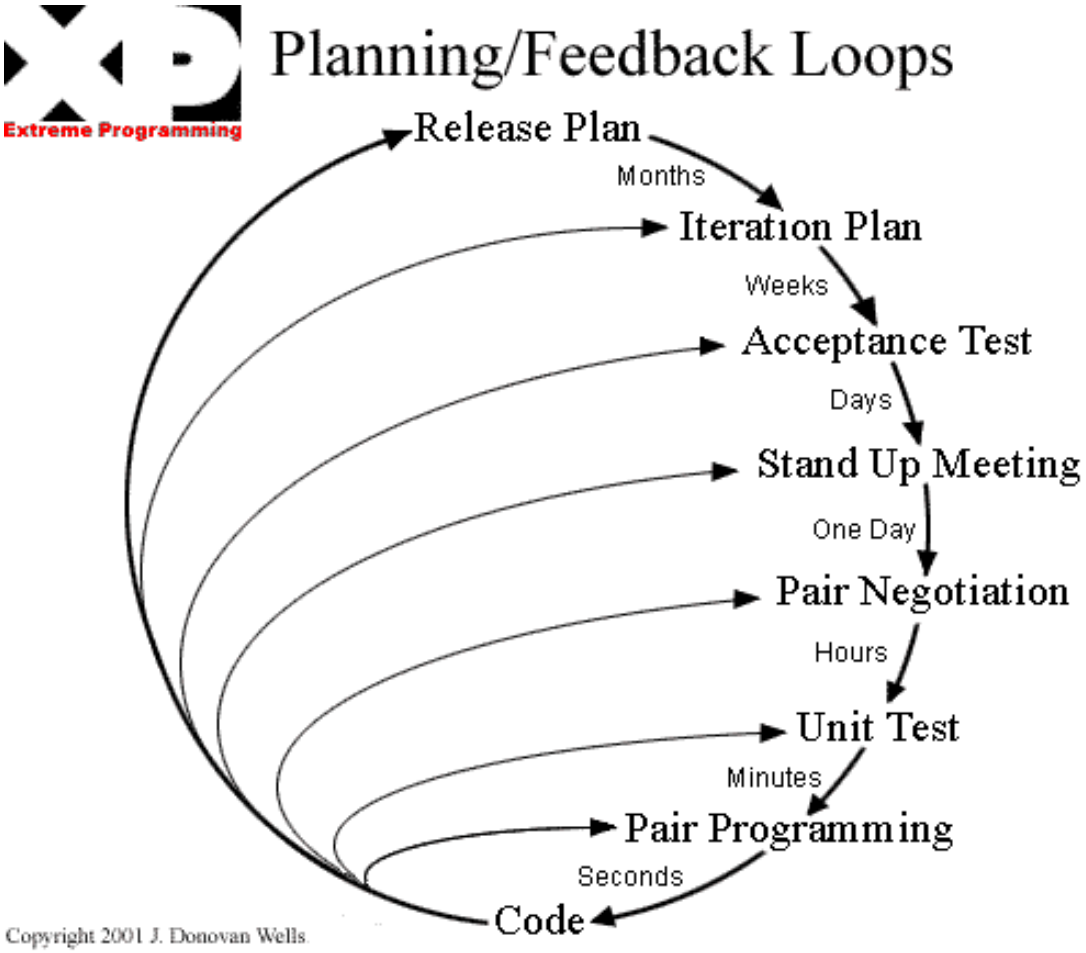


**Figura 5.2. Ciclos en eXtreme Programming**

## 5.3. Aplicación de *eXtreme Programming* en ONess

Se ha optado por la programación extrema XP ya que ésta se ajusta mejor a las características del proyecto.

Más concretamente, se cumplen las recomendaciones para emplear XP en un proyecto [Wells03] [Ferrer03]:

- Interés sincero por todas las partes en que el proyecto tenga éxito.
- El equipo de trabajo es pequeño.
- No existe un contrato fijo previo especificando tiempo, recursos y alcance.
- El equipo dispone de una formación elevada (esa es la finalidad) y capacidad de aprender.

- El proyecto tiene un riesgo alto en cuanto a lo innovador de la tecnología.

También se ha tenido en cuenta el éxito que ha tenido *extreme programming* en proyectos open source, cuyas características hacen que se adapte especialmente bien esta metodología.

Como es evidente no todas las prácticas son aplicables al presente proyecto, por lo que a continuación se especifica para cada una de ellas su aplicación en ONess:

1. Planificación incremental

   Se ha decidido realizar cuatro *releases* con sus respectivas iteraciones. Cada una de ellas proporcionando un valor de negocio claro, a grosso modo serán:

   - *party*: gestión de clientes y proveedores
   - *user*: gestión de usuarios
   - *inventory*: gestión de inventario
   - *order*: gestión de pedidos, albaranes y facturas

   En el Capítulo 7, *Planificacion de las entregas* se detallará cada una de estas iteraciones.

   En cuanto a la valoración de los costes de usar distintas opciones tecnológicas dado el fin principalmente de innovación no se ha utilizado para realizar una comparación, si bien se ha sido consciente de que el uso de nuevas tecnologías supone un coste importante de aprendizaje. Este coste se convierte en beneficios posteriores, como son un inferior tiempo de desarrollo al añadir nueva funcionalidad una vez que el núcleo del sistema ha sido realizado y, aunque no relacionado específicamente con este proyecto, una gran experiencia para su autor.

2. Testing

   Una de las principales aportaciones de esta metodología es el concepto de desarrollo dirigido por tests (*Test Driven Development*). Los tests son realizados a priori con el fin de prevenir errores, no de solucionarlos. Esto confiere una gran calidad al software resultante. Los tests son ejecutados automáticamente cada día para asegurar que el sucesivo desarrollo del sistema no afecta a su estabilidad. Dado que los tests son creados a priori fallarán hasta que la funcionalidad esté implementada, en el proceso que comúnmente se llama *rojo a verde*. En caso de que se encuentren fallos no detectados previamente, es requisito escribir un test que falle para posteriormente solucionar el problema.

   La automatización de los tests proporciona una gran seguridad, por ejemplo en las pruebas del sistema con distintos gestores de base de datos se tiene la certeza de que si los tests se

ejecutan correctamente la compatibilidad existe.

3. Programación en parejas

   Dado el impuesto carácter individual del proyecto esta práctica no ha sido aplicada.

4. Refactorización

   En las sucesivas iteraciones ha sido necesario refactorizar partes del núcleo del sistema y este proceso ha sido realmente sencillo, a lo que ha contribuido en gran parte la cantidad de tests realizados.

5. Diseño simple

   El diseño se ha mantenido sencillo, desde luego pasando los tests, sin código duplicado y con un mínimo de código gracias al núcleo del sistema que proporciona un gran dinamismo y evita la necesidad de implementación de operaciones repetitivas.

6. Propiedad colectiva del código

   A pesar de que en este proyecto no ha existido un grupo de desarrolladores como tal, gracias a su licencia *open source* y al sitio web que se ha puesto a disposición de toda la comunidad en [Sourceforge], se ha llevado a la práctica esta propiedad colectiva, donde cualquier desarrollador puede examinar el código y realizar las modificaciones que consideren pertinentes, pudiendo revertirse estas contribuciones en el proyecto.

7. Integración continua

   La integración es permanente gracias a los tests de integración y la automatización con la herramienta Maven.

8. Cliente en el equipo

   Aunque esto estrictamente no se realiza dada la naturaleza del proyecto, la experiencia del autor en el ámbito empresarial tratado hace que se tengan en todo momento presentes los intereses y visión de negocio.

9. *Releases* pequeñas

   Se ha seguido esta práctica, liberando nuevas versiones según la funcionalidad se ha ido implementando.

10. Semanas de 40 horas

    No es aplicable a este proyecto.

11. Estándares de codificación

    Se ha seguido el estándar de codificación Java, definido por Sun en [JavaCodeConventions].

12. Uso de Metáforas

    Se ha utilizado el vocabulario de negocio en inglés por su mayor alcance a nivel mundial. Para facilitar la comunicación con los posibles usuarios se han establecido mecanismos como listas de correo y un *weblog* para facilitar la lectura de las noticias que genera el proyecto.

# Parte III. Ciclo de vida

# Tabla de contenidos

# Capítulo 6. Exploración

Tal y como se ha definido en el ciclo de vida de *eXtreme Programming* la primera fase es la exploración, fase en la que se plantean las historias de usuario (*user stories*) al mismo tiempo que el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo.

## 6.1. Historias de usuario (*user stories*)

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del software, lo que equivaldría a los casos de uso en el proceso unificado.

Se describen brevemente las características que el sistema debe tener desde la perspectiva del cliente, en este caso hay tres grupos de funcionalidades bien diferenciadas: gestión de clientes y proveedores, gestión de inventario y gestión de compras y ventas.

### 6.1.1. Funcionalidad general

Como requisitos generales del sistema se consideran principalmente auditoría, seguridad y la accesibilidad desde dispositivos móviles.

#### 6.1.1.1. Auditoría

Los cambios realizados al sistema por los usuarios deben ser auditados. Para todas las altas, modificaciones y eliminaciones de datos se debe conocer el qué, quién y cuándo se han realizado.

#### 6.1.1.2. Autenticación y autorización

Existirán varios roles de usuarios, y cada uno de ellos sólo podrá acceder a un subconjunto de los datos y las operaciones. Existirá un usuario administrador que tendrá acceso completo a la aplicación.

#### 6.1.1.3. Accesibilidad desde dispositivos móviles

El sistema debe ser accesible desde cualquier localización utilizando tanto ordenadores portátiles como dispositivos móviles, principalmente PDAs. La localización puede ser tanto el propio local de la empresa como un lugar exterior.

### 6.1.2. Gestión de contactos: clientes y proveedores

En este apartado se tratará el concepto de contacto (*party*), persona u organización que es la otra parte en una relación comercial.

Cada uno de estos contactos puede tiene como atributo un nombre interno, que es el utilizado comúnmente para referirse a él, y unos atributos según sea:

- persona
  - título (Sr., Sra.)
  - nombre
  - apellidos
- organización
  - nombre oficial

Además de esto pueden tener información para contactar con ellos, como pueden ser:

- direcciones postales (destinatario, dirección, ciudad, código postal, provincia, país)
- números de teléfono (país, código de área, teléfono, extensión)
- direcciones de correo electrónico (e-mail)
- direcciones web (url)

Cada uno de ellos con el propósito de esa información de contacto, por ejemplo

- correo electrónico del departamento de ventas
- dirección del almacén
- dirección de las oficinas
- etc.

#### 6.1.2.1. Creación de contactos

Para la creación de un contacto el usuario debe seleccionar si es una persona o una organización, y según su elección introducir los atributos correspondientes.

#### 6.1.2.2. Visualización de contactos

Al usuario se le muestran los atributos correspondientes al tipo de contacto y la lista con las informaciones de contacto.

#### 6.1.2.3. Modificación de contactos

El usuario podrá cambiar los atributos de un contacto.

#### 6.1.2.4. Borrado de contactos

Cuando se considere que ya no son de relevancia los clientes o proveedores podrán ser borrados.

#### 6.1.2.5. Búsqueda de contactos

Se podrán realizar búsquedas por cualquiera de los atributos, tanto de personas como de organizaciones.

#### 6.1.2.6. Añadir información de contacto a un contacto

Desde la visualización de un contacto se podrá añadir información de contacto, seleccionando el tipo de información (dirección postal, número de teléfono, correo electrónico o dirección web) se permitirá que el usuario introduzca los atributos correspondientes.

#### 6.1.2.7. Visualización de información de contacto

Se mostrará el tipo de información, sus atributos y los propósitos de ella.

#### 6.1.2.8. Modificación de información de contacto

Se podrán cambiar los atributos de cualquier tipo de información de contacto.

#### 6.1.2.9. Borrado de información de contacto

Cuando ya no sea relevante para al negocio se podrá borrar la información de contacto.

### 6.1.3. Gestión de inventario

La gestión de inventario se centra en las operaciones con modelos (*models*) y productos (*products*). Esta es una de las partes más complicadas en cualquier empresa, ya que se tiene que tratar con productos procedentes de diversos proveedores, que llaman de distintas formas a la

misma cosa, que tienen unos procesos muy distintos unos a otros,...

En el negocio textil esto es más complicado si cabe, donde apenas se utilizan los pocos estándares que existen, no se utiliza la misma nomenclatura para los tallajes ni los colores y el uso de códigos de barras es inferior al 50%, por lo que aproximaciones que habitualmente son usadas en otro tipo de negocios no son aplicables.

Volviendo a modelos y productos, los modelos representan un diseño concreto, del que pueden existir distintas tallas y colores. Los modelos son exclusivos de cada proveedor, es decir dos proveedores nunca hacen un mismo modelo, siempre hay alguna diferencia como puede ser la calidad del tejido. Los modelos tienen los siguientes atributos:

- código de modelo del proveedor
- nombre, asignado por la empresa, que normalmente será el código asignado por el proveedor, pero que podría ser cambiado si ya existiese ese código anteriormente o el proveedor no asignara ninguno.
- descripción
- año
- temporada, principalmente
    - permanente
    - primavera-verano
    - otoño-invierno
- tallaje
    - recién nacido (000, 00, ..., 30, 36)
    - niño (1, 2, ..., 8, 10)
    - niño mayor (4, 6, ..., 16, 18)
    - caballero (48, 50, ..., 58, 60)
    - señora (5, 6, 7, 8)

Cada modelo tiene productos, que son la representación de cada talla y color, con su SKU (*Stock Keeping Unit*) que es el identificador que se utilizará para referirse a ellos dentro de la empresa, y la posibilidad de tener un número ilimitado de códigos de barras, para evitar los problemas causados por la posible gestión incorrecta de los proveedores.

Se considera la posibilidad de que la empresa tenga más de un almacén, si bien esto no es lo habitual, con lo que en cada uno de ellos podría existir stock de los productos.

También se considera que pueden existir varias tarifas, cada una con un precio distinto para cada producto.

#### 6.1.3.1. Creación de modelos

Par crear un modelo el usuario debe seleccionar el proveedor al que pertenece, introducir el código del modelo del proveedor, el nombre, la descripción, el año, la temporada, el tallaje al que pertenece y seleccionar los colores en los que lo habrá. Automáticamente se crearán los productos correspondientes a cada talla del tallaje y a cada color seleccionado.

#### 6.1.3.2. Visualización de modelos

La visualización de un modelo consistirá en mostrar sus atributos así como cada uno de sus productos, con su talla, color, stock en cada uno de los almacenes, precio en cada tarifa y sus códigos de barras.

#### 6.1.3.3. Modificación de modelos

Una vez creado, los atributos del modelo pueden ser modificados.

#### 6.1.3.4. Borrado de modelos

Borrar un modelo equivale a darlo de baja porque ni se dispone de stock ni se dispondrá en el futuro.

#### 6.1.3.5. Búsqueda de modelos

Se podrá buscar un modelo por cualquiera de sus atributos, cuyo resultado será una lista de modelos desde la que poder ir a la visualización de cada uno de ellos.

#### 6.1.3.6. Creación y modificación de precios

Desde la visualización de un modelo se podrá ir a la creación/modificación de sus precios, que consistirá en poder asignar un precio o cambiar el ya existente para cada una de las tarifas y para todas las tallas.

### 6.1.4. Gestión de compras y ventas

Dentro de esta funcionalidad se engloba el proceso de creación de

- pedidos (*orders*)
- albaranes (*delivery dockets*)
- facturas (*invoices*)

tanto para compras como para ventas.

El proceso de ventas comienza cuando un representante visita a un cliente en su lugar de negocio. Este cliente escoge una serie de productos y cantidades que se formalizan en un pedido.

Posteriormente los pedidos son servidos según se considere oportuno. Se podrán servir en su totalidad o parcialmente, quedando pendientes las cantidades no servidas para posteriores ocasiones, pudiendo también servir en un único envío mercancía de más de un pedido. Se generará un albarán con la mercancía enviada, corresponda a uno o varios pedidos, es decir si se sirve mercancía de dos pedidos en un único envío se generará un único albarán.

Finalmente se generarán las facturas a partir de los albaranes, pudiendo una factura incluir varios albaranes. Normalmente estas facturas se incluyen en los envíos por lo que sólo se incluirá el albarán correspondiente.

El proceso de compra es exactamente el mismo.

#### 6.1.4.1. Creación de pedidos

A partir de un cliente o proveedor se podrá crear un nuevo pedido, seleccionando si se trata de un pedido de compra o de venta. El único dato necesario será la fecha del pedido, que normalmente será el mismo día, y se podrán anotar cualquier tipo de comentarios.

#### 6.1.4.2. Añadir productos a un pedido

Se podrán buscar los productos para añadirlos al pedido en curso, en la cantidad deseada.

#### 6.1.4.3. Visualización de pedidos

Al visualizar un pedido se mostrarán los datos del cliente o proveedor y la lista de productos.

#### 6.1.4.4. Modificación de pedidos

La fecha o los comentarios del pedido pueden ser modificados, así como la lista de productos, pudiendo añadir, quitar o modificar las cantidades de éstos.

#### 6.1.4.5. Borrado de pedidos

Si un pedido es anulado se podrá borrar.

#### 6.1.4.6. Listado de pedidos pendientes

La lista de pedidos pendientes debe estar accesible, preferiblemente con un indicador sobre la posibilidad de ser servido parcial o totalmente.

#### 6.1.4.7. Creación de albaranes

Crear un albarán implica seleccionar de entre los pedidos pendientes de un cliente o proveedor la mercancía que es servida, seleccionando las cantidades que se servirán de cada producto, normalmente el total.

#### 6.1.4.8. Creación de facturas

Un proceso equivalente al de creación de albaranes, seleccionando de entre los albaranes no facturados de un cliente o proveedor aquellos que se facturarán, siempre en su totalidad.

## 6.2. Herramientas

En el desarrollo de este proyecto se han utilizado las siguientes herramientas en cada ámbito. Todo el software utilizado es gratuito y tan sólo la herramienta Poseidon no es *open source*.

### 6.2.1. Desarrollo

- Java™ 2 SDK, Standard Edition 1.4.2-04 Sun Microsystems
- Jikes java compiler 1.20 para una compilación más rápida.
- Poseidon for UML Community Edition 2.4.1 para la realización de diagramas UML
- Eclipse IDE SDK 3.0 con los siguientes plugins:
  - Spring IDE plugin for Eclipse 1.0.1 para la integración de Spring Framework en el IDE
  - Mevenide 0.2.1 para la integración con Maven
  - Sysdeo Eclipse Tomcat Launcher plugin 2.2 para la integración con Tomcat permitiendo la depuración de código en el servidor.
- CVS (Concurrent Versions System) CVSNT 2.0.2

- PuTTY 2004-08-08 cliente SSH

### 6.2.2. Gestión del proyecto

Se utiliza Maven 1.0 para la gestión integral del proyecto, todo lo necesario está explícitamente especificado en el descriptor de proyecto de Maven.

### 6.2.3. Ejecución

El software requerido para la ejecución del sistema es:

1. Un servidor de aplicaciones web (Tomcat, Jetty,...)
2. Un sistema gestor de base de datos relacional (MySQL, PostgreSQL, Oracle, ...)
3. Un navegador web (Internet Explorer, Mozilla, Netscape Navigator, Opera, ...)

Este es el software que ha sido utilizado para la ejecución y los tests del sistema para cada uno de los roles anteriores.

1. Servidor de aplicaciones web
   - Jakarta Tomcat 5.0.25
2. Base de datos
   - MySQL 4.0.13-Max bajo Windows
   - MySQL 4.0.15-Max bajo Linux
   - PostgreSQL 7.4.5 bajo Cygwin en Windows
   - MS SQL 2000
   - HSQLDB 1.7.2, esta versión **NO** es compatible con el software dado que no soporta niveles de aislamiento entre transacciones.
3. Navegadores web
   - Internet Explorer 6
   - Mozilla Firefox 0.8
   - PalmOS Garnet Simulator 5.4 + PalmSource Web Browser 2.0 SDK for Palm OS

Garnet para comprobar los resultados en una PDA con el sistema operativo PalmOS

## 6.3. Tecnologías

Lo novedoso de las tecnologías usadas es uno de los puntos fuertes de este proyecto. Todas ellas son tecnologías Java *open source*. El aprendizaje y familiarización ha ocupado la mayor parte del tiempo del proyecto.

### 6.3.1. Maven

Maven es una herramienta de gestión de información de proyectos. Maven está basado en el concepto de un modelo de objetos del proyecto POM (*Project Object Model*) en el que todos los productos (*artifacts*) generados por Maven son el resultado de consultar un modelo de proyecto bien definido. Compilaciones, documentación, métricas sobre el código fuente y un innumerable número de informes son todos controlados por el POM.

Maven tiene muchos objetivos, pero resumiendo Maven intenta hacer la vida del desarrollador sencilla proporcionando una estructura de proyecto bien definida, unos procesos de desarrollo bien definidos a seguir, y una documentación coherente que mantiene a los desarrolladores y clientes informados sobre lo que ocurre en el proyecto. Maven aligera en gran cantidad lo que la mayoría de desarrolladores consideran trabajo pesado y aburrido y les permite proseguir con la tarea. Esto es esencial en proyectos *open source* donde no hay mucha gente dedicada a la tarea de documentar y propagar la información crítica sobre el proyecto que es necesaria para atraer potenciales nuevos desarrolladores y clientes.

La ambición de Maven es hacer que el desarrollo interno del proyecto sea altamente manejable con la esperanza de proporcionar más tiempo para el desarrollo entre proyectos. Se puede llamar polinización entre proyectos o compartir el conocimiento sobre el desarrollo del proyecto.

Características:

- El modelo de objetos del proyecto POM es la base de cómo Maven trabaja. El desarrollo y gestión del modelo está controlado desde el modelo del proyecto.
- Un único conjunto de métodos son utilizados para todos los proyectos que se gestionan. Ya no hay necesidad de estar al tanto de innumerables sistemas de compilación. Cuando las mejoras se hacen en Maven todos los usuarios se benefician.
- Integración con Gump, una herramienta usada en el proyecto Jakarta para ayudar a los proyectos a mantener compatibilidad con versiones anteriores.

- Publicación del sitio web basado en el POM. Una vez el POM es exacto los desarrolladores pueden publicar fácilmente el contenido del proyecto, incluyendo la documentación personalizada más el amplio conjunto de documentación generada por Maven a partir del código fuente.
- Publicación de distribuciones basada en el POM.
- Maven alenta el uso de un repositorio central de librerías, utilizando un mecanismo que permite descargar automáticamente aquellas necesarias en el proyecto, lo que permite a los usuarios de Maven reutilizar librerías entre proyectos y facilita la comunicación entre proyectos para asegurar que la compatibilidad entre distintas versiones es correctamente tratada.
- Guías para la correcta disposición de los directorios. Maven contiene documentación sobre como disponer los directorios de forma que una vez es aprendida se puede ver fácilmente cualquier otro proyecto que use Maven.

### 6.3.2. Programación Orientada a Aspectos (AOP)

La Programación Orientada a Aspectos, más conocida como AOP por su nombre en inglés *Aspect Oriented Programming*, es un modelo de programación que aborda un problema específico: capturar las partes de un sistema que los modelos de programación habituales obligan a que estén repartidos a lo largo de distintos módulos del sistema. Estos fragmentos que afectan a distintos módulos son llamados **aspectos** y los problemas que solucionan, problemas cruzados (*crosscutting concerns*).

Usando un lenguaje que soporte AOP, podemos capturar estas dependencias en módulos individuales, obteniendo un sistema independiente de ellos y podemos utilizarlos o no sin tocar el código del sistema básico, preservando la integridad de las operaciones básicas.

Los principales campos de aplicación de la AOP son:

- rastreo de la ejecución (*tracing*)
- medida de tiempos y optimización (*profiling*)
- pruebas (*testing*)

Los principales frameworks existentes para AOP en Java son

- JBoss AOP
- Nanning

- Aspectwerkz
- AspectJ

De todos ellos el último, **AspectJ**, es el más maduro y con mayor número de características.

AOP ha experimentado un gran éxito y otros frameworks como Spring Framework han introducido características de la programación orientada a aspectos de una manera más sencilla y en muchos casos transparente para el desarrollador.

### 6.3.3. Programación Orientada a Atributos

Este paradigma se integra dentro de la programación orientada a objetos, y se basa en la utilización de atributos en el código fuente para especificar comportamientos de las clases que normalmente se describen en ficheros de configuración. El objetivo es evitar la dispersión de la información en varios puntos como código fuente y distintos ficheros de configuración.

El origen de este paradigma está en la especificación EJB, que requiere un gran número de ficheros de configuración, clases e interfaces en muchas ocasiones redundantes. Con el fin de facilitar el desarrollo de sistemas EJB se creó XDoclet, que interpretaba una serie de etiquetas JavaDoc especiales y generaba los ficheros redundantes a partir de ellas.

Este paradigma ha sufrido un gran auge a partir de su utilización en el entorno .NET y en estos momentos el mundo Java se ha percatado de su importancia, incluyendo soporte en la nueva especificación Java 5.

### 6.3.4. AspectJ

AspectJ es una extensión orientada a aspectos del lenguaje de programación Java. Permite la aplicación de aspectos a clases Java para la solución de los problemas cruzados. Un compilador de AspectJ produce ficheros *class* conformes a la especificación del *bytecode* de Java, permitiendo que sean ejecutados en cualquier máquina virtual de Java. Al utilizar Java como lenguaje base, AspectJ proporciona todos los beneficios de Java y hace que sea sencillo que los desarrolladores Java entiendan el lenguaje AspectJ.

AspectJ está formado por dos partes: la especificación del lenguaje y la implementación del lenguaje. La parte de especificación del lenguaje define el lenguaje en el que se escribe el código; se implementa la funcionalidad principal con el lenguaje Java y se utilizan las extensiones proporcionadas por AspectJ para implementar el entrelazado (*weaving*) de los problemas cruzados. La parte de implementación de lenguaje proporciona herramientas para compilar, depurar e integrar AspectJ con los entornos de desarrollo más populares.

En AspectJ la implementación de las reglas de entrelazado (*weaving*) por el compilador es llamado atajo (*crosscutting*). Las reglas de entrelazado atajan hacia múltiples módulos de manera sistemática con el objetivo de modularizar los problemas cruzados. AspectJ define dos tipos de *crosscutting*, *crosscutting* estático y *crosscutting* dinámico

- *crosscutting* dinámico

  Es el entrelazado de nuevo comportamiento durante la ejecución de un programa. La mayoría de *crosscutting* que ocurre en AspectJ es dinámico. El *crosscutting* dinámico aumenta o incluso reemplaza el flujo de ejecución principal del programa de una manera que afecta a los distintos módulos, por lo tanto modificando el comportamiento del sistema. Por ejemplo, se puede especificar que una acción determinada sea ejecutada antes de la ejecución de ciertos métodos o manejadores de excepciones en un conjunto de clases tan sólo especificando en un módulo separado los puntos de entrelazado y la acción a realizar cuando se alcanzan esos puntos.

- *crosscutting* estático

  Es el entrelazado de modificaciones en la estructura estática (clases, interfaces y aspectos) del sistema. La aplicación principal del *crosscutting* estático es dar soporte a la implementación del *crosscutting* dinámico. Por ejemplo, se pueden añadir nuevos datos y métodos a clases e interfaces para definir estados comportamientos específicos a nivel de clase que pueden ser usados en el *crosscutting* dinámico. Otro uso del *crosscutting* estático es declarar advertencias y errores en tiempo de compilación a través de múltiples módulos.

### 6.3.5. Spring Framework

Spring es un framework de aplicaciones Java/J2EE desarrollado por los autores de [JohnsonHoeller04], [TateGehtland04] y [Johnson02], basado en las ideas expuestas en éste último.

Spring proporciona:

- Una potente gestión de configuración basada en JavaBeans, aplicando los principios de Inversión de Control (*IoC*). Esto hace que la configuración de aplicaciones sea rápida y sencilla. Ya no es necesario tener *singletons* ni ficheros de configuración, una aproximación consistente y elegante. Esta factoría de *beans* puede ser usada en cualquier entorno, desde *applets* hasta contenedores J2EE. Estas definiciones de *beans* se realizan en lo que se llama el contexto de aplicación.

- Una capa genérica de abstracción para la gestión de transacciones, permitiendo gestores de

transacción enchufables (*pluggables*), y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel. Se incluyen estrategias genéricas para JTA y un único JDBC DataSource. En contraste con el JTA simple o EJB CMT, el soporte de transacciones de Spring no está atado a entornos J2EE.

- Una capa de abstracción JDBC que ofrece una significativa jerarquía de excepciones (evitando la necesidad de obtener de SQLException los códigos que cada gestor de base de datos asigna a los errores), simplifica el manejo de errores, y reduce considerablemente la cantidad de código necesario.

- Integración con Hibernate, JDO e iBatis SQL Maps en términos de soporte a implementaciones DAO y estrategias con transacciones. Especial soporte a Hibernate añadiendo convenientes características de *IoC,* y solucionando muchos de los comunes problemas de integración de Hibernate. Todo ello cumpliendo con las transacciones genéricas de Spring y la jerarquía de excepciones DAO.

- Funcionalidad AOP, totalmente integrada en la gestión de configuración de Spring. Se puede aplicar AOP a cualquier objeto gestionado por Spring, añadiendo aspectos como gestión de transacciones declarativa. Con Spring se puede tener gestión de transacciones declarativa sin EJB, incluso sin JTA, si se utiliza una única base de datos en un contenedor web sin soporte JTA.

- Un framework MVC (*Model-View-Controller*), construido sobre el núcleo de Spring. Este framework es altamente configurable vía interfaces y permite el uso de múltiples tecnologías para la capa vista como pueden ser JSP, Velocity, Tiles, iText o POI. De cualquier manera una capa modelo realizada con Spring puede ser fácilmente utilizada con una capa web basada en cualquier otro framework MVC, como Struts, WebWork o Tapestry.

Toda esta funcionalidad puede usarse en cualquier servidor J2EE, y la mayoría de ella ni siquiera requiere su uso. El objetivo central de Spring es permitir que objetos de negocio y de acceso a datos sean reusables, no atados a servicios J2EE específicos. Estos objetos pueden ser reutilizados tanto en entornos J2EE (web o EJB), aplicaciones standalone, entornos de pruebas,... sin ningún problema.

La arquitectura en capas de Spring Figura 6.1, “Spring: arquitectura en capas” ofrece cantidad de flexibilidad. Toda la funcionalidad está construida sobre los niveles inferiores. Por ejemplo se puede utilizar la gestión de configuración basada en JavaBeans sin utilizar el framework MVC o el soporte AOP.

**Figura 6.1. Spring: arquitectura en capas**

### 6.3.6. Acegi Security System for Spring

Acegi Security proporciona servicios de seguridad dentro de Spring Framework. Aunque no forma parte directa de Spring está íntimamente ligado con éste.

Proporciona una serie de características clave:

- Facilidad de uso
- *Single Sign On*

  Proporciona un sistema de autenticación a través del cual los usuarios pueden autenticarse y acceder a múltiples aplicaciones a través de un único punto de entrada. Para ello utiliza el servicio de autenticación CAS (*Central Authentication Service*) desarrollado por la Universidad de Yale [CAS], con el que una aplicación utilizando Acegi Security puede participar en un entorno *single sign on* a nivel de toda la empresa. Ya no es necesario que cada aplicación tenga su propia base de datos de autenticación, ni tampoco existe la restricción de que sólo se pueda utilizar dentro del mismo servidor de aplicaciones. Otras características avanzadas que proporciona son soporte para proxy y refresco forzado de logins.

- Integración completa en Spring

  Utiliza los mecanismos de configuración de Spring

- Seguridad a nivel de instancia de objetos del dominio

  En muchas aplicaciones es deseable definir listas de control de acceso (*Access Control Lists* o *ACLs*) para instancias de objetos del dominio individuales. Proporciona un completo paquete ACL con características que incluyen máscaras de bits, herencia de permisos, un repositorio utilizando JDBC, caché y un diseño *pluggable* utilizando interfaces.

- Configuración no intrusiva

  La totalidad del sistema de seguridad puede funcionar en una aplicación web utilizando los filtros que proporciona. No hay necesidad de hacer cambios especiales o añadir librarías al contenedor de Servlets o EJB.

- Integración opcional con los contenedores

  Las características de autenticación y autorización que proporcionan los contenedores Servlet o EJB pueden ser usadas utilizando los adaptadores que se incluyen. Actualmente existe soporte para los principales contenedores: Catalina (Tomcat), Jetty, JBoss y Resin

- Mantiene los objetos libres de código de seguridad

  Muchas aplicaciones necesitan proteger datos a nivel de objeto basándose en cualquier combinación de parámetros (usuario, hora del día, autoridad del usuario, método que es llamado, parámetros del método invocado,...). Acegi da esta flexibilidad sin necesidad de añadir código a los objetos de negocio.

- Protección de peticiones HTTP

  Además de proteger los objetos, el proyecto también permite proteger las peticiones HTTP. Ya no es necesario depender de restricciones de seguridad definidas en el fichero web.xml. Lo mejor de todo es que las peticiones HTTP pueden ser protegidas por una serie de expresiones regulares o expresiones de paths como las utilizadas por Ant, así como autenticación, autorización y gestores de reemplazo de credenciales para la ejecución como otro usuario, todo ello totalmente *pluggable*.

- Seguridad del canal

  El sistema de seguridad puede redirigir automáticamente las peticiones a un canal de transmisión adecuado. Comúnmente esto se aplica para asegurar que las páginas seguras estarán sólo disponibles sobre HTTPS, y las páginas públicas sobre HTTP, aunque es suficientemente flexible para soportar cualquier tipo de requisitos de "canal". También soporta combinaciones de puertos no usuales y gestores de decisión de transporte *pluggables*.

- Soporta autenticación HTTP BASIC

  Esta autenticación es adecuada para aquellas aplicaciones que prefieren una simple ventana de login del navegador en lugar de un formulario de login. Acegi Security puede procesar directamente peticiones de autenticación HTTP BASIC siguiendo el RFC 1945.

- Librería de etiquetas

  Proporciona una librería de etiquetas que puede ser utilizada en JSPs para garantizar que contenido protegido como enlaces y mensajes son únicamente mostrados a usuarios que poseen los permisos adecuados.

- Configuración mediante el contexto de aplicación de Spring o basada en atributos

  Permite seleccionar el método utilizado para la configuración de seguridad del entorno, tanto a través del contexto de aplicación de Spring o utilizando atributos en el código fuente utilizando Jakarta Commons Attributes.

- Distintos métodos de almacenamiento de la información de autenticación

Acegi incluye la posibilidad de obtener los usuarios y permisos utilizando ficheros XML, fuentes de datos JDBC o implementando un interfaz DAO para obtener la información de cualquier otro lugar.

- Soporte para eventos

  Utilizando los servicios para eventos que ofrece Spring, se pueden configurar receptores propios para eventos como login, contraseña incorrecta y cuenta deshabilitada. Esto permite la implementación de sistemas de auditoría y bloqueo de cuentas, totalmente desacoplados del código de Acegi Security.

- Caché

  Utilizando EHCache o otra implementación propia se puede hacer caché de la información de autenticación, evitando que la base de datos o cualquier otro tipo de fuente de información no sea consultado repetidamente.

### 6.3.7. Hibernate

Hibernate es un potente mapeador objeto/relacional y servicio de consultas para Java. Es la solución ORM (*Object-Relational Mapping*) más popular en el mundo Java.

Hibernate permite desarrollar clases persistentes a partir de clases comunes, incluyendo asociación, herencia, polimorfismo, composición y colecciones de objetos. El lenguaje de consultas de Hibernate HQL (*Hibernate Query Language*), diseñado como una mínima extensión orientada a objetos de SQL, proporciona un puente elegante entre los mundos objetual y relacional. Hibernate también permite expresar consultas utilizando SQL nativo o consultas basadas en criterios.

Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones J2EE y contenedores web, y por supuesto también puede utilizarse en aplicaciones *standalone*.

Características clave:

- Persistencia transparente

  Hibernate puede operar proporcionando persistencia de una manera transparente para el desarrollador.

- Modelo de programación natural

  Hibernate soporta el paradigma de orientación a objetos de una manera natural: herencia,

polimorfismo, composición y el framework de colecciones de Java.

- Soporte para modelos de objetos con una granularidad muy fina

  Permite una gran variedad de mapeos para colecciones y objetos dependientes.

- Sin necesidad de mejorar el código compilado (*bytecode*)

  No es necesaria la generación de código ni el procesamiento del *bytecode* en el proceso de compilación.

- Escalabilidad extrema

  Hibernate posee un alto rendimiento, tiene una caché de dos niveles y puede ser usado en un cluster. Permite inicialización perezosa (*lazy*) de objetos y colecciones.

- Lenguaje de consultas HQL

  Este lenguaje proporciona una independencia del SQL de cada base de datos, tanto para el almacenamiento de objetos como para su recuperación.

- Soporte para transacciones de aplicación

  Hibernate soporta transacciones largas (aquellas que requieren la interacción con el usuario durante su ejecución) y gestiona la política *optimistic locking* automáticamente.

- Generación automática de claves primarias

  Soporta los diversos tipos de generación de identificadores que proporcionan los sistemas gestores de bases de datos (secuencias, columnas autoincrementales,...) así como generación independiente de la base de datos, incluyendo identificadores asignados por la aplicación o claves compuestas.

### 6.3.8. XDoclet

XDoclet es un motor de generación de código a partir de plantillas basado en el paradigma de programación orientada a atributos, lo que significa que genera código a partir de metadatos (atributos) añadidos a los ficheros fuente Java.

Para ello utiliza tags JavaDoc especiales, que obtendrá analizando el código fuente, y que servirán para generar otros ficheros como descriptores XML o más código fuente a partir de unas plantillas y los atributos, proporcionando una serie de ventajas:

- Reducir el trabajo superfluo

XDoclet permite generar el código redundante a partir de un único punto de información, por ejemplo interfaces EJB a partir de las clases.

- Simplificar el desarrollo de aplicaciones J2EE

  Permite al desarrollador implementar el *enterprise bean* y XDoclet genera interfaces, objetos valor, formularios de struts,...

- Soporte para los servidores y herramientas más usados

  Permite generar ficheros de configuración para los principales servidores de aplicaciones (JBoss, BEA WebLogic, IBM WebSphere, Oracle IAS, Orion, Borland, MacroMedia JRun, Jonas, Pramati, Sybase EAServer) y herramientas (Castor, Hibernate, varias implementaciones JDO, Struts, WebWork, MockObjects).

- Extensibilidad

  Su diseño modular permite implementar módulos propios para la generación de código a partir de plantillas.

### 6.3.9. Struts, JSP, JSTL, Tiles, struts-menu, Validator, CSS

Struts es un framework MVC desarrollado dentro de la *Apache Software Foundation* que proporciona soporte a la creación de las capas vista y controlador de aplicaciones web basadas en la arquitectura *Model2*. Está basado en tecnologías estándar como Servlets, JavaBeans y XML.

Struts proporciona un controlador que se integra con una vista realizada con páginas JSP, incluyendo JSTL y Java Server Faces, entre otros. Este controloador evita la creación de servlets y delega en acciones creadas por el desarrollador, simplificando sobremanera el desarrollo de aplicaciones web.

En cuanto a la capa vista, Struts permite utilizar entre otras tecnologías páginas JSP para la realización del interfaz. Para facilitar las tareas comunes en la creación de esta capa existen una serie de tecnologías que se integran con Struts:

- Struts taglib, una librería de etiquetas que proporciona numerosa funcionalidad evitando el escribir código Java en las páginas JSP.

- JSTL, la librería de etiquetas estándar de Java que añade funcionalidades a la librería de tags de Struts y sustituye alguna de las ya presentes.

- Tiles, una extensión que permite dividir las páginas JSP en componentes reusables para la

construcción del interfaz.

- Struts Validator, proporciona validación de los formularios basándose en reglas fácilmente configurables.
- Struts Menú, un proyecto que basándose en un fichero de configuración genera vistosos menús.
- CSS, hojas de estilo en cascada, que permite la realización de interfaces web de mayor calidad y separa mejor la presentación de los datos.

### 6.3.10. JUnit, DBUnit, JMock, StrutsTestCase

JUnit es el standard *de facto* para realizar los tests de unidad de las aplicaciones Java.

Para completar la funcionalidad ofrecida por JUnit otros proyectos han surgido:

- DBUnit

  Proyecto que permite realizar tests que involucren una base de datos, insertando datos antes de la ejecución de los tests y comprobando los datos tras ella. También permite importar y exportar los datos desde y hacia ficheros xml o xls (Excel).

- JMock

  Librería que permite realizar tests utilizando objetos simulados (*mock objects*) dinámicos. El objetivo es aislar los objetos que se testean sustituyendo los objetos con los que se relacionan por objetos simulados en los que podemos definir su estado y los resultados de los métodos.

- StrutsTestCase

  Extensión de JUnit que proporciona facilidades para comprobar código basado en Struts utilizando dos posibles implementaciones, una basada en *mock objects* que no requiere la ejecución dentro de un contenedor de aplicaciones, y otra basada en el proyecto Cactus para la ejecución dentro de un contenedor.

### 6.3.11. Jakarta Commons

Jakarta *commons* es un conjunto de proyectos desarrollado dentro de la *Apache Software Foundation* (ASF) que está formado por un gran número de utilidades que facilitan las tareas comunes en la creación de aplicaciones.

## 6.4. Prototipo

Tras la selección de tecnologías se establece la arquitectura general del sistema que se puede ver en la Figura 6.2, “Arquitectura del sistema”, con las tres capas definidas por el patrón MVC (*Model-View-Controller*).

**Figura 6.2. Arquitectura del sistema**

Esta arquitectura se desarrollará durante la primera iteración a la vez que se realiza el prototipo, donde se comprobará su adecuación al desarrollo del sistema.

# Capítulo 7. Planificacion de las entregas

En esta fase se establece la prioridad de cada historia de usuario así como una estimación del esfuerzo necesario de cada una de ellas con el fin de determinar un cronograma de entregas.

Las estimaciones de esfuerzo asociado a la implementación de las historias se establecen utilizando como medida el punto. Un punto, equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. Por otra parte, se mantiene un registro de la "velocidad" de desarrollo, establecida en puntos por iteración, basándose principalmente en la suma de puntos correspondientes a las historias de usuario que fueron terminadas en la última iteración.

La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias se pueden implementar antes de una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuántos puntos se pueden completar. Al planificar según alcance del sistema, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación.

## 7.1. Estimaciones de esfuerzo

### 7.1.1. Funcionalidad general

**Tabla 7.1. Funcionalidad general**

| | |
|---|---|
| Funcionalidad común | 2 |
| Auditoría | 2 |
| Autenticación y autorización | 2 |
| Accesibilidad desde dispositivos móviles | 1 |

### 7.1.2. Gestión de contactos: clientes y proveedores

**Tabla 7.2. Gestión de contactos**

| | |
|---|---|
| Creación de contactos | 1 |

| | |
|---|---|
| Visualización, modificación, eliminación y búsqueda de contactos | 2 |
| Añadir información de contacto a un contacto | 1 |
| Visualización, modificación y eliminación de información de contacto | 1 |

### 7.1.3. Gestión de inventario

**Tabla 7.3. Gestión de inventario**

| | |
|---|---|
| Creación de modelos y productos | 1 |
| Visualización, modificación, eliminación y búsqueda de modelos | 1 |
| Creación y modificación de precios | 1 |

### 7.1.4. Gestión de compras y ventas

**Tabla 7.4. Gestión de compras y ventas**

| | |
|---|---|
| Creación de pedidos y añadir productos a un pedido | 1 |
| Visualización, modificación, eliminación y listado de pedidos | 1 |
| Creación de albaranes y facturas | 1 |

## 7.2. Planificación

Partiendo de las historias de usuario anteriores se realiza una planificación en cuatro iteraciones basándose en el tiempo y procurando agrupar la funcionalidad relacionada en la misma iteración.

En la Tabla 7.5, “Fechas de entrega” pueden verse las fechas reales en las que cada versión ha sido entregada a través del sitio web [Sourceforge]. El motivo por el que se crean nuevas versiones de alguno de los módulos puede ser cambios desde la última versión o simplemente

actualización de las dependencias en otros módulos cuando una nueva versión de éstos últimos es entregada, para evitar posibles problemas. Con estos datos y teniendo en cuenta que el desarrollo como tal comenzó a finales del mes de abril se obtienen los datos reales de duración de cada iteración tal y como se muestran en cada apartado.

**Tabla 7.5. Fechas de entrega**

| | **2ª semana julio** | **4ª semana julio** | **4ª semana agosto** | **3ª semana septiembre** |
|---|---|---|---|---|
| common | 0.1 | | 0.2 | 0.3 |
| common-maven | | 0.1-0.2[a] | 0.3 | 0.4 |
| common-model | 0.1 | 0.2 | 0.3 | 0.4 |
| common-webapp-controller | 0.1 | 0.2 | 0.3 | 0.4 |
| common-webapp-taglib | | 0.1 | | |
| common-webapp-view | | 0.1 | 0.2 | 0.3 |
| party-model | 0.1 | 0.2 | 0.3 | 0.4 |
| party-webapp | 0.1 | 0.2 | 0.3 | 0.4 |
| user-model | | 0.1 | 0.2 | 0.3 |
| user-webapp | | 0.1 | 0.2 | 0.3 |
| inventory-model | | | 0.1 | 0.2 |
| inventory-webapp | | | 0.1 | 0.2 |
| order-model | | | | 0.1 |
| order-webapp | | | | 0.1 |

[a]0.2 es una versión de mantenimiento que soluciona problemas de integración de la versión anterior

## 7.2.1. Primera iteración: Prototipo

En esta primera iteración se creará un prototipo con el que se comprobará la adecuación de la tecnología escogida y se intentará crear la mayor parte de la base de la arquitectura del sistema, que será encapsulada dentro de los módulos *common*. No se implementará una funcionalidad muy extensa tan sólo un mínimo para tener cuanto antes una *demo* que poder mostrar en el sitio web y así atraer posibles usuarios.

Como se puede comprobar, la duración real de esta iteración ha superado en dos semanas el

tiempo estimado. Este retraso ha sido debido principalmente a la curva de aprendizaje de la tecnología usada.

**Tabla 7.6. Historias primera iteración**

| | |
|---|---|
| Funcionalidad común | 2 |
| Auditoría | 2 |
| Creación de contactos | 1 |
| Visualización, modificación, eliminación y búsqueda de contactos | 2 |
| **ESTIMACIÓN INICIAL** | **7** |
| **REAL** | **9** |

### 7.2.2. Segunda iteración: Autenticación y autorización, finalización de la gestión de contactos

En una segunda iteración se añadirá la funcionalidad necesaria para gestionar la autenticación y autorización de los usuarios, y se completará la gestión de contactos comenzada en la iteración anterior.

La duración real de la iteración ha sido muy breve gracias a que el núcleo del sistema ya había sido realizado en la iteración anterior con todo el esfuerzo de integración de tecnologías. Una vez hecho esto, la gestión de autenticación y autorización ha sido sencilla de integrar en el sistema, así como también ha sido breve la implementación de la gestión de contactos.

**Tabla 7.7. Historias segunda iteración**

| | |
|---|---|
| Añadir información de contacto a un contacto | 1 |
| Visualización, modificación y eliminación de información de contacto | 1 |
| Autenticación y autorización | 2 |
| **ESTIMACIÓN INICIAL** | **4** |
| **REAL** | **2** |

### 7.2.3. Tercera iteración: Gestión de inventario y accesibilidad desde dispositivos móviles

En esta tercera iteración se añadirá la funcionalidad relativa a la gestión de inventario. También se hará que el sistema sea accesible desde dispositivos móviles.

A pesar de problemas surgidos que han requerido la realización de cambios en el núcleo del sistema, la duración de la iteración no ha aumentado, lo que indica que se ha sobreestimado el esfuerzo de las historias de usuario.

**Tabla 7.8. Historias tercera iteración**

| | |
|---|---|
| Creación de modelos y productos | 1 |
| Visualización, modificación, eliminación y búsqueda de modelos | 1 |
| Creación y modificación de precios | 1 |
| Accesibilidad desde dispositivos móviles | 1 |
| **ESTIMACIÓN INICIAL** | **4** |
| **REAL** | **4** |

### 7.2.4. Cuarta iteración: Gestión de compras y ventas

La cuarta iteración conllevará la finalización del sistema tras la implementación de las historias correspondientes a la gestión de compras y ventas.

El tiempo real de desarrollo han sido dos semanas ya que la primera semana de septiembre no se ha dedicado al desarrollo del proyecto, reduciéndose lo estimado tal y como se podía extraer de anteriores planificaciones donde se ha constatado que la implementación de nueva funcionalidad es un proceso bastante ágil.

**Tabla 7.9. Historias cuarta iteración**

| | |
|---|---|
| Creación de pedidos y añadir productos a un pedido | 1 |
| Visualización, modificación, eliminación y listado de pedidos | 1 |
| Creación de albaranes y facturas | 1 |

| | |
|---|---|
| **ESTIMACIÓN INICIAL** | **3** |
| **REAL** | **2** |

# Capítulo 8. Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El Plan de Entrega está compuesto por iteraciones de no más de tres semanas. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, se puede variar con el fin de maximizar el valor de negocio. Al final de la última iteración el sistema estará listo para entrar en producción.

Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior.

## 8.1. Primera iteración

En esta iteración se contempla la realización del prototipo que además de servir para evaluar la tecnología también establecerá la arquitectura base del sistema.

Las historias de usuario a abordar se pueden ver en la Tabla 8.1, “Historias primera iteración”.

**Tabla 8.1. Historias primera iteración**

| | |
|---|---|
| Funcionalidad común | 2 |
| Auditoría | 2 |
| Creación de contactos | 1 |
| Visualización, modificación, eliminación y búsqueda de contactos | 2 |
| **ESTIMACIÓN INICIAL** | **7** |
| **REAL** | **9** |

### 8.1.1. Estructura de directorios y repositorio de código fuente

Para comenzar el desarrollo es necesario configurar una estructura de directorios adecuada que facilite las tareas. Utilizando Maven esto no supone un gran problema ya que para cada proyecto se tiene una estructura bien definida, con lo que nos debemos centrar en la correspondencia entre nuestro proyecto y sus módulos en proyectos Maven, teniendo en cuenta

que una de las principales filosofías de Maven es la de que cada proyecto genere un único *artefacto*, que es así como llama a los ficheros jar, war,...

Con el objeto de fomentar la reusabilidad el código se separará en módulos basándose en funcionalidad, y dentro de cada módulo se creará un subproyecto para la capa modelo y otro para el interfaz web, que agrupará las capas controlador y vista.

### 8.1.2. Auditoría

Para auditar los cambios que se realizan a los datos del sistema es necesario saber una serie de datos según el tipo de operación que se realice:

- creación

  Cuando un nuevo objeto es creado, se debe saber quién lo ha creado y cuándo.

- modificación

  Cuando se modifica un objeto es necesario saber quién lo ha modificado y cuándo, y además guardar la versión anterior por si fuera necesario restaurarlo.

- borrado

  Al borrar un objeto se necesita saber quién lo ha borrado y cuándo, y también se debe guardar por si se necesitara restaurar.

De todo lo anterior se deduce que una buena forma de permitir la auditoría es guardar la fecha de creación de un objeto cuando se crea, la de eliminación cuando se borra y el usuario que lo ha creado y borrado respectivamente. A partir de la fecha de borrado o usuario que lo ha borrado, se podrá determinar si un objeto está eliminado o no.

El problema surge con la modificación de objetos, con lo que la aproximación anterior se completa de la siguiente manera: cuando un objeto es modificado se marcará como borrado con fecha de eliminación y autor, y se creará un nuevo objeto que tendrá las mismas propiedades que el objeto modificado, con fecha de creación y autor.

Con esta aproximación lo que se consideraría identificador del objeto pasaría a representar una única versión del objeto, siendo necesaria otra propiedad que nos permita relacionar entre sí todas las versiones del objeto, lo que se llamará código (`code`). Todas las versiones de un mismo objeto tendrán el mismo código pero cada una con su identificador único, y se podrá seguir la evolución desde su creación hasta su borrado mediante este código y las fechas de creación y eliminación.

Al tratar con información temporal deben tratarse dos dimensiones [Fowler]:

- momento en el que la información cambia (*transaction time* o *record time*)
- momento en el que la información es efectiva (*valid time* o *actual time*)

En el caso de la auditoría la definición aplicable es la primera, y se utilizará el nombre *transaction time* para referirse al rango de fechas que comienza en el momento en el que un dato es introducido en el sistema y finaliza cuando este dato deja de ser relevante y es borrado. Este rango de fechas será limitado cuando el objeto esté eliminado o esa versión haya sido modificada, y tendrá su final en el infinito mientras el objeto siga siendo válido.

## 8.1.3. Funcionalidad común

En el módulo *common* se irán añadiendo las funcionalidades que se prevé serán utilizadas por más de un módulo para facilitar su reusabilidad.

Dado que la url en la que se encuentra el sitio web es http://oness.sourceforge.net (o también http://oness.sf.net), el paquete base de Java será **net.sf.oness**, escogiéndose la segunda url por ser más breve y más común entre otros proyectos alojados en Sourceforge utilizar sf en lugar de sourceforge para la nomenclatura de los paquetes. A partir de este paquete base se crearán uno por cada módulo, en este caso, el paquete base para la funcionalidad común será **net.sf.oness.common**.

En una primera aproximación se diferencian claramente tres tipos de funcionalidades comunes según a la capa de la aplicación que afectarán:

- all

  Funcionalidad que afectará a la totalidad del sistema.

- model

  Funcionalidad que afectará tan sólo a la capa modelo del sistema.

- webapp

  Funcionalidad que afectará sólo a las capas vista y controlador de las aplicaciones web.

Cada submódulo de los anteriores se corresponderá con un paquete, **net.sf.oness.common.all**, **net.sf.oness.common.model** y **net.sf.oness.common.webapp** respectivamente.

### 8.1.3.1. Funcionalidad común a la totalidad del sistema

Es necesario extraer ciertas tareas repetitivas que serán comunes a gran parte de las clases Java. Se diferencian dos objetivos claros: facilitar el depurado (*logging* y *tracing*) e implementación de métodos definidos en el contrato Java (`toString`, `equals`, `hashCode` y `clone`).

#### 8.1.3.1.1. Facilitar el depurado: *logging* y *tracing*

Este es uno de los campos típicos donde se aplica la Programación Orientada a Aspectos [Laddad03], que nos proporciona los mecanismos para implementar esta funcionalidad de una manera sencilla. El objetivo es mostrar las llamadas a los métodos con sus parámetros, la finalización de estas llamadas y las excepciones que ocurren durante la ejecución.

La tecnología que mejor se adapta es AspectJ, desarrollando un aspecto que se aplique a todas las clases y que pueda deshabilitarse tras la fase de desarrollo de una forma sencilla. También debe permitirse el filtrado de los mensajes de logging a nivel de clase, paquete o tipo de mensaje (entrada de método, salida de método o lanzamiento de excepciones). Para ello este aspecto deberá utilizar un framework como Apache Commons Logging que proporciona una capa de abstracción sobre otros sistemas de logging que pueden ser configurados declarativamente, como Log4J, que será el usado normalmente por su sencillez de configuración y potencia de uso

El aspecto desarrollado `LoggingAspect` es totalmente genérico y aplicable a cualquier sistema. Algunas consideraciones que ha sido necesario tener en cuenta son:

- precedencia

  el aspecto de logging debe tener mayor precedencia que otros aspectos para que los mensajes de entrada en un método se ejecuten antes que cualquier otro aspecto que se ejecute antes de la ejecución de un método, y los mensajes de salida después de los otros aspectos que se ejecuten después de la ejecución de un método.

- evitar la recursividad infinita

  no se debe hacer logging del propio aspecto así como tampoco de las ejecuciones que tienen lugar dentro de los métodos llamados por él, principalmente el método `toString` que puede ser llamado implícitamente por la máquina virtual al hacer operaciones con strings y objetos.

#### 8.1.3.1.2. Implementación de métodos definidos en el contrato Java: `toString`, `equals`, `hashCode` y `clone`

La implementación de estos métodos aunque no es siempre imprescindible suele ser deseable, y supone una tarea tediosa que hay que realizar en cada clase que se implementa. Para facilitar o

incluso evitar esta tarea se crea una clase que hará de raíz de la jerarquía de clases siempre que sea posible, implementando de manera genérica los métodos `toString`, `equals`, `hashCode` y `clone` haciendo uso extensivo de los metadatos que proporciona Java a través del API *reflection*. Para facilitar este trabajo ya existen proyectos de la ASF, principalmente los proyectos *commons-lang* y *commons-beanutils*.

- `toString`, `equals` y `hashCode` son implementados utilizando *commons-lang* de manera que se utilizan todas sus propiedades para mostrar de manera homogénea el estado del objeto, hacer comparaciones entre objetos o calcular su código hash, respectivamente.
- `clone` es implementado usando *commons-beanutils*, que proporciona una manera sencilla de clonar superficialmente un objeto copiando todas sus propiedades.

#### 8.1.3.2. Funcionalidad común a la capa modelo

### 8.1.3.2.1. Soporte a la auditoría

Como soporte a la auditoría se creará el paquete *auditing*, creándose en primer lugar un interfaz `Auditable` que deben implementar aquellos objetos que pretendan ser auditados. Tal y como se mencionó en Sección 8.1.2, “Auditoría”, aquellos objetos auditables deben tener:

- identificador (`id`)
- código (`code`)
- fechas de transacción (`transactionTime`)
- creado-por (`createdBy`)
- borrado-por (`deletedBy`)

Junto con este interfaz se crea una implementación por defecto `AbstractAuditableObject` de la que pueden heredar otras clases.

### 8.1.3.2.2. Objetos del dominio (*Business Objects*)

Se seguirá en patrón *Business Object* [AlurCrupiMalks03] para reflejar el modelo conceptual del dominio. Estos objetos del dominio serán persistidos mediante Hibernate, por lo que es necesario afinar los métodos definidos en `BaseObject`, lo que se hará en la clase `AbstractBusinessObject`, debido a que Hibernate puede utilizar una caché perezosa para cargar las colecciones, con lo que los métodos `toString`, `equals` y `hashCode` definidos en la superclase se sobrecargarán para ignorar colecciones. El método `clone` debe ser también sobrecargado para evitar que una colección sea referenciada por dos objetos distintos,

restricción que impone Hibernate y que se soluciona creando una nueva colección pero con los mismos objetos.

#### 8.1.3.2.3. Soporte para tests

Para facilitar la implementación de los tests, principalmente en cuestiones relacionadas con la integración del framework Spring, se han realizado unas clases utilidad.

- `SpringApplicationContext`

  Una clase que proporciona métodos estáticos para acceder al contexto de aplicación de Spring desde los tests.

- `SpringDatabaseExport`

  Una utilidad para exportar una base de datos a partir de una fuente de datos definida en el contexto de aplicación de Spring.

- `SpringDatabaseTestCase`

  Una clase base para los tests que integra DBUnit con Spring, permitiendo que DBUnit utilice una fuente de datos definida en el contexto de aplicación de Spring.

#### 8.1.3.2.4. Objetos de Acceso a Datos (DAOs)

Para acceder a los datos se utilizará el patrón DAO (*Data Access Object* [AlurCrupiMalks03]) que ocultará la implementación utilizada por si se diera la necesidad de cambiarla en el futuro. Dado que la persistencia será gestionada en principio con Hibernate, que permite el acceso a gran cantidad de metadatos, se intentará realizar un DAO genérico que proporcione los servicios de persistencia para cualquier clase sin necesidad de escribir más líneas de código.

Se usará también el soporte que proporciona Spring para la integración con Hibernate, concretamente la clase `HibernateDaoSupport`. Se implementará un DAO genérico, `HibernateDao`, con las siguientes operaciones típicas:

- `findById` obtener un objeto a partir de su identificador
- `create` crear un objeto persistente
- `update` actualizar un objeto persistente
- `delete` no se implementará por no ser necesario ya que los datos no se borrarán nunca de la base de datos por cuestiones de auditabilidad.

Dado que el DAO será genérico es necesario pasarle la clase a la que proporcionará persistencia

en el momento de su creación.

Para ocultar detalles de implementación se crearán dos interfaces:

- `FinderDao`

  Este interfaz contendrá aquellos métodos de sólo lectura que permitirán realizar búsquedas en las base de datos, comenzando con `find`, un método que implementa la búsqueda por criterio, es decir a partir de un objeto con algunas propiedades encuentra todos aquellos con propiedades coincidentes. Utilizando el patrón *Value List Handler* [AlurCrupiMalks03] no devolverá todos los objetos que cumplan el criterio sino sólo un número determinado, en forma de `PaginatedList`, que representa un rango de objetos dentro de una lista.

- `Dao`

  Interfaz de conveniencia que extiende `AuditingDao` (ver Sección 8.1.3.2.5, “Auditoría”) y `FinderDao`.

La configuración dentro del contexto de aplicación de Spring se puede ver en el fichero `applicationContext-oness-common-model.xml`, donde está centralizada para todo el sistema.

**Ejemplo 8.1. Configuración general de Hibernate**

```
<!-- Session Factory -->
<bean id="sessionFactory"
    class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
    <property name="mappingResources">
        <ref bean="mappingResources" />
    </property>
    <property name="hibernateProperties">
        <ref bean="hibernateProperties" />
    </property>
</bean>

<!-- Transaction manager for a single Hibernate SessionFactory
     (alternative to JTA) -->
<bean id="transactionManager"
      class="org.springframework.orm.hibernate.HibernateTransactionManager">
    <property name="sessionFactory">
        <ref local="sessionFactory" />
    </property>
</bean>
```

En este fichero se establece la configuración general de Hibernate aplicable a todos los

módulos, el `sessionFactory` y el gestor de transacciones. Se delega en los contextos de aplicación de cada módulo para la configuración concreta, así se centraliza y se evita la replicación de estos parámetros. En concreto se espera que cada módulo defina un bean llamado `mappingResources` que por ejemplo podría ser algo como esto:

**Ejemplo 8.2. Configuración del fichero de mapeo de Hibernate de Party**

```
    <bean id="mappingResources" class="java.util.ArrayList">
        <constructor-arg>
            <list>
                <value>net/sf/oness/party/model/party/bo/Party.hbm.xml</value>
            </list>
        </constructor-arg>
    </bean>
```

También se evita la definición de los beans `dataSource` e `hibernateProperties` para permitir utilizar indistintamente un *dataSource* local usando por ejemplo commons-DBCP o uno obtenido mediante JNDI, útil en servidores de aplicaciones J2EE. Tan sólo es necesario definir en tiempo de ejecución cuál de los siguientes ficheros se añaden al contexto de aplicación:

- `applicationContext-oness-common-model-dataSource-dbcp.xml`

  Útil para la ejecución de los tests fuera de un servidor J2EE. Las propiedades concretas, definidas como `${...}`, no están en este fichero, sino en `dataSource.properties` que se obtiene del *classpath*, de esta forma no es necesario modificarlo, sino que simplemente se puede añadir otro fichero con el mismo nombre en el *classpath* antes que él y tendrá mayor precedencia. En caso de que alguna de las propiedades exista como propiedades del sistema también tendrán mayor precedencia, lo que es de gran utilidad a la hora de ejecutar los tests con Maven, ya que sin necesidad de cambiar o crear ningún fichero se podrán ejecutar los tests en cualquier sistema gestor de bases de datos en cualquier máquina.

  **Ejemplo 8.3. Configuración de la fuente de datos DBCP**

```
<!-- Get datasource properties from file -->
<bean id="propertyConfigurer"
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="location">
        <value>classpath:dataSource.properties</value>
    </property>
    <!-- Override properties in file with system properties -->
    <property name="systemPropertiesModeName">
        <value>SYSTEM_PROPERTIES_MODE_OVERRIDE</value>
    </property>
</bean>
```

```
<!-- DBCP Basic datasource -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
    <property name="driverClassName">
        <value>${dataSource.driverClassName}</value>
    </property>
    <property name="url">
        <value>${dataSource.url}</value>
    </property>
    <property name="username">
        <value>${dataSource.username}</value>
    </property>
    <property name="password">
        <value>${dataSource.password}</value>
    </property>
</bean>

<!-- Hibernate properties -->
<bean id="hibernateProperties"
      class="net.sf.oness.common.model.dao.hibernate.HibernateProperties">
    <constructor-arg>
        <props>
            <prop key="hibernate.dialect">${hibernate.dialect}</prop>
            <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
            <prop key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
        </props>
    </constructor-arg>
</bean>
```

`dataSource.properties` utilizadas normalmente:

**Ejemplo 8.4. Propiedades de la fuente de datos DBCP**

```
hibernate.show_sql=false
hibernate.hbm2ddl.auto=create

# MySQL
dataSource.username=❶
dataSource.password=❷
dataSource.url=jdbc:mysql:///test❸
dataSource.driverClassName=com.mysql.jdbc.Driver❹
hibernate.dialect=net.sf.hibernate.dialect.MySQLDialect❺
```

❶ Nombre de usuario en la base de datos

❷ Contraseña en la base de datos

❸ URL de la base de datos, depende de cada gestor

❹ Nombre de la clase controladora JBDC, depende de cada gestor

❺ Dialecto del gestor de la base de datos, proporcionado por Hibernate

- `applicationContext-oness-common-model-dataSource-jndi.xml`

  Útil para la ejecución en servidores J2EE como Tomcat. Permite ejecutar la aplicación sin necesidad de modificar el fichero war distribuido, ya que la configuración de los parámetros necesarios se realiza en el contenedor.

**Ejemplo 8.5. Configuración de la fuente de datos JNDI**

```
<!-- JNDI DataSource for J2EE environments -->
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
        <value>java:comp/env/jdbc/oness</value>
    </property>
</bean>

<!-- Hibernate Dialect -->
<bean id="hibernateDialect" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
        <value>java:comp/env/net.sf.oness.common.model.hibernateDialect</value>
    </property>
</bean>

<!-- Hibernate properties -->
<bean id="hibernateProperties"
      class="net.sf.oness.common.model.dao.hibernate.HibernateProperties">
    <constructor-arg>
        <props>
            <prop key="hibernate.show_sql">false</prop>
            <prop key="hibernate.hbm2ddl.auto">create</prop>
        </props>
    </constructor-arg>
    <property name="hibernateDialect"><ref local='hibernateDialect'/></property>
</bean>
```

La configuración concreta para el servidor Tomcat, utilizando el módulo *party* como ejemplo, se especifica a continuación. Tan sólo algunas propiedades (en negrita) necesitan ser modificadas para utilizar otro sistema gestor de base de datos u otra máquina como servidor.

**Ejemplo 8.6. Configuración de la aplicación web party-webapp en Tomcat utilizando JNDI**

```
<!--

    To setup this context copy this file to the specified
    directory and change it to match your needs.
    You may rename it if you want.

    o Tomcat 4: TOMCAT_HOME/webapps
    o Tomcat 5: TOMCAT_HOME/conf/Catalina/localhost
```

```
-->

<Context path="/oness-party-webapp"
    docBase="${catalina.home}/webapps/oness-party-webapp.war"
    debug="99"
    reloadable="true">

    <!--
        To use a global jndi datasource uncomment the <ResourceLink> tag
        and move the other entries to your server.xml
        under <GlobalNamingResources>
    -->
    <!--
    <ResourceLink name="jdbc/oness"
        global="jdbc/oness"
        type="javax.sql.DataSource"/>
    -->

    <Environment description="Hibernate dialect"
        name="net.sf.oness.common.model.hibernateDialect"
        value="net.sf.hibernate.dialect.MySQLDialect"
        type="java.lang.String"/>

    <Resource name="jdbc/oness"
        auth="Container"
        type="javax.sql.DataSource"/>

    <ResourceParams name="jdbc/oness">
        <parameter>
            <name>factory</name>
            <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </parameter>
        <!-- Maximum number of dB connections in pool. Make sure you
             configure your mysqld max_connections large enough to handle
             all of your db connections. Set to 0 for no limit.
             -->
        <parameter>
            <name>maxActive</name>
            <value>100</value>
        </parameter>
        <!-- Maximum number of idle dB connections to retain in pool.
             Set to 0 for no limit.
             -->
        <parameter>
            <name>maxIdle</name>
            <value>30</value>
        </parameter>
        <!-- Maximum time to wait for a dB connection to become available
             in ms, in this example 10 seconds. An Exception is thrown if
             this timeout is exceeded.  Set to -1 to wait indefinitely.
             -->
        <parameter>
            <name>maxWait</name>
            <value>10000</value>
        </parameter>
        <!-- MySQL dB username and password for dB connections  -->
        <parameter>
            <name>username</name>
```

```
            <value></value>
        </parameter>
        <parameter>
            <name>password</name>
            <value></value>
        </parameter>
        <!-- Class name for JDBC driver -->
        <parameter>
            <name>driverClassName</name>
            <value>com.mysql.jdbc.Driver</value>
        </parameter>
        <!-- Autocommit setting.  This setting is required to make
             Hibernate work.  Or you can remove calls to commit(). -->
        <parameter>
            <name>defaultAutoCommit</name>
            <value>false</value>
        </parameter>
        <!-- The JDBC connection url for connecting to your MySQL dB.
             The autoReconnect=true argument to the url makes sure that the
             mm.mysql JDBC Driver will automatically reconnect if mysqld closed the
             connection.  mysqld by default closes idle connections after 8 hours.
             -->
        <parameter>
            <name>url</name>
            <value>jdbc:mysql://localhost/test</value>
        </parameter>
        <!-- Recover abandoned connections -->
        <parameter>
            <name>removeAbandoned</name>
            <value>true</value>
        </parameter>
        <!-- Set the number of seconds a dB connection has been idle
             before it is considered abandoned.
             -->
        <parameter>
            <name>removeAbandonedTimeout</name>
            <value>60</value>
        </parameter>
        <!-- Log a stack trace of the code which abandoned the dB
             connection resources.
             -->
        <parameter>
            <name>logAbandoned</name>
            <value>true</value>
        </parameter>
    </ResourceParams>

</Context>
```

### 8.1.3.2.5. Auditoría

En este apartado se implementará lo definido en Sección 8.1.2, “Auditoría”, creando un aspecto que intercepte las llamadas a los DAOs de la siguiente manera:

- `create`

Antes de crear el objeto persistente se establecería el inicio de la fecha de transacción al momento actual y el final a infinito.

- `delete`

  Se sustituye la llamada al método `delete` del DAO por una llamada a `update` tras poner el final de la fecha de transacción al momento actual.

- `update`

  Se sustituye la llamada al método `update` del DAO por dos llamadas, una a `update` para marcar como borrado el objeto anterior y otra a `create` para crear la nueva versión.

En principio este aspecto se ha implementado con AspectJ, pero posteriormente se ha implementado mediante el soporte AOP de Spring dado que la primera opción requería utilizar el compilador de AspectJ y la segunda permite una integración más fácil dado que no modifica el *bytecode* java, sino que utiliza proxies dinámicos.

- `AuditableDaoAdvisor`

  Este es el aspecto que interceptará las llamadas a los DAOs y las delegará a `AuditingDaoHelper`.

- `AuditingDaoHelper`

  En esta clase se implementan las reglas mencionadas anteriormente que se ejecutarán cada vez que se intercepte una llamada.

Para dar soporte se añaden los siguientes interfaces:

- `AuditableDao`

  Interfaz con los métodos necesarios que debe implementar un DAO para permitir la auditoría (`findById`, `create` y `update`).

- `AuditingDao`

  Este interfaz extiende `AuditableDao` y añade el método `delete`.

Para facilitar la utilización de fechas y rangos de fechas se crean las clases:

- `Date`

  Encapsula la clase `Calendar` de Java truncando su precisión normalmente al segundo ya que

no es necesario más ni es soportada por muchas bases de datos, y añade nuevas funcionalidades. Se definen dos constantes `MIN_VALUE` y `MAX_VALUE` que representan respectivamente el infinito negativo y positivo.

- `DateRange`

  Está formada por un `Date` inicial y uno final y añade métodos para trabajar con rangos de fechas (inclusión, duración,...).

Para poder persistir estas clases con Hibernate como componentes y poder reutilizarlas es necesario implementar unos interfaces:

- `DateType`

  Persiste `Date` implementando `UserType`.

- `DateRangeType`

  Persiste `DateRange` implementando `CompositeUserType`. Principalmente se transforman las constantes `MIN_VALUE` y `MAX_VALUE` a valores `null` en SQL.

La configuración del aspecto se realiza dentro del contexto de aplicación de Spring en el fichero `applicationContext-oness-common-model.xml` para que esté disponible para todos los módulos del sistema.

**Ejemplo 8.7. Configuración del aspecto de auditoría en Spring**

```
<!-- AuditingDao AOP -->
<bean id="autoProxyCreator"
  class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator">
</bean>
<bean id="auditingDaoAdvisor"
      class="net.sf.oness.common.model.dao.AuditableDaoAdvisor">
</bean>
```

Se utiliza la funcionalidad de creación automática de proxies de Spring para que el aspecto se aplique automáticamente a todos aquellos *beans* definidos en el contexto de aplicación que cumplan el contrato establecido por el método `matches` del aspecto.

**Ejemplo 8.8. Método matches del aspecto de auditoría**

```
/**
 * @see org.springframework.aop.MethodMatcher#matches(java.lang.reflect.Method,
 *      java.lang.Class)
```

```
 */
public boolean matches(Method m, Class targetClass) {
    return AuditableDao.class.isAssignableFrom(targetClass);
}
```

#### 8.1.3.2.6. Utilidades

Detro de este paquete se puede encontrar una implementación del interfaz `List` del api *collections* de Java que proporciona una manera de tratar listas paginadas, aquellas en las que normalmente sólo se accede a una pequeña porción, tal y como se define en el patrón *ValueListHandler* [AlurCrupiMalks03].

También se incluye `DatabasePopulator`, una clase que configurada desde el contexto de aplicación de Spring permite importar datos de un fichero de DBUnit tanto en formato xml como xls (Excel) en la base de datos. Se utilizará para insertar los datos de ejemplo para las demostraciones.

### 8.1.3.3. Funcionalidad común a las capas controlador y vista de las aplicaciones web

Para no tener que implementar una acción de Struts para cada acción de la vista se utilizarán `DispatchActions` que permitirán utilizar una única clase para varias urls de peticiones, agrupando también la funcionalidad relacionada. Para facilitar la utilización se creará una acción `AutoDispatchAction` que basándose en la url de la petición escogerá automáticamente el método que debe ser llamado, por ejemplo urls del tipo create* o update* provocarán llamadas a los métodos `create` o `update` respectivamente.

Para integrar Spring con Struts y poder acceder al contexto de aplicación de donde se obtendrá la correspondiente fachada de la capa modelo se añade también la clase `SpringActionSupport` que permite obtener de manera sencilla cualquier objeto del contexto de aplicación de Spring.

Los tests de unidad deben comprobar porciones de código lo más pequeñas posible y de forma aislada, por lo que será necesario utilizar *mock objects* que sustituyan la fachada de la capa modelo para aislarla de esta forma de la capa controlador. Se utilizará con este fin el *framework* JMock, que proporciona una solución basada en *mocks* dinámicos mucho más eficiente en el desarrollo que soluciones estáticas que requieren la generación de código fuente. Por otro lado también es necesario probar las acciones de Struts aisladamente sin necesidad de ejecutarlas en un contenedor, para lo que se ajusta el proyecto StrutsTestCase, una extensión de JUnit. Para poder integrar ambas aproximaciones, JMock y StrutsTestCase, es necesario introducir una nueva clase `JMockStrutsTestCase` de la que extenderán los tests concretos.

Posteriormente este módulo ha sido movido a *common-webapp-controller* al crear los nuevos

módulos *common-webapp-view* y *common-webapp-taglib* en la segunda iteración.

## 8.1.4. Creación, visualización, modificación, eliminación y búsqueda de contactos

### 8.1.4.1. Modelo

Comenzamos con los objetos del dominio que surgen en las historias relativas a los contactos y se realiza el diagrama UML que se puede ver en la Figura 8.1, “Creación, visualización, modificación, eliminación y búsqueda de contactos” para el diseño de la capa modelo.

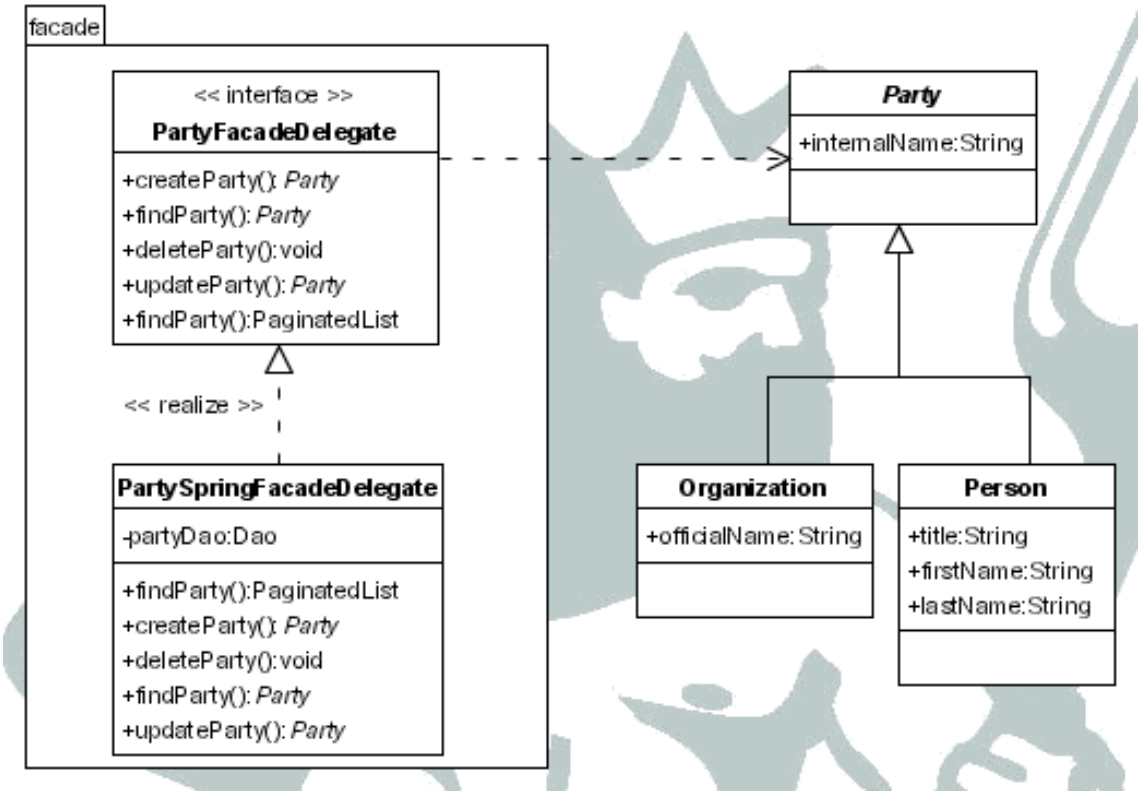


**Figura 8.1. Creación, visualización, modificación, eliminación y búsqueda de contactos**

Como se puede ver se tratan los conceptos de

- party: representa un contacto
- person: aquellos contactos que son personas
  - title: título (Sr., Sra.)
  - firstName: nombre

    - lastName: apellidos

- organization: aquellos contactos que son organizaciones

    - officialName: nombre oficial

Como interfaz de la capa modelo hacia capas superiores dentro de la arquitectura MVC se crea la fachada (*façade*) o servicio *PartyFacade* para ocultar los detalles de implementación de la capa modelo, desempeñando el papel de los patrones *SessionFacade* y *BusinessDelegate* [AlurCrupiMalks03].

Para gestionar la persistencia de estos objetos se etiquetan los ficheros fuente con atributos XDoclet que serán procesados mediante el plugin XDoclet de Maven para generar los ficheros de configuración de Hibernate que especifican para cada clase cómo se mapearán esos objetos en el sistema gestor de bases de datos.

Por ejemplo para la clase Party

**Ejemplo 8.9. Definición de atributos de persistencia en la clase Party**

```
/**
 * Value object for Party.
 *
 * The subclasses MUST override the getType() method
 *
 * @hibernate.class table="party"
 *
 * @hibernate.discriminator column="type"
 *
 * @author Carlos Sanchez
 */
public class Party extends AbstractBusinessObject {
```

La línea en negrita especifica que los objetos de esta clase serán persistidos en la tabla *party* en la base de datos.

**Ejemplo 8.10. Definición de atributos de persistencia en los métodos de la clase Party**

```
    /**
     * @hibernate.property
     *
     * @return String
     */
    public String getInternalName() {
        return this.internalName;
    }
```

```
    public void setInternalName(String internalName) {
        this.internalName = internalName;
    }
```

En este caso la línea en negrita define que la propiedad `internalName` es persistente, con lo que la tabla en la base de datos tendrá una columna para guardar sus valores.

En cuanto a la fachada será necesaria una implementación del interfaz `PartyFacadeDelegate` que utilice Spring, `PartySpringFacadeDelegate`. Es una buena práctica crear un interfaz para que los detalles de implementación queden ocultos a las otras capas. Entre estos detalles de implementación se encuentra una nueva propiedad, `partyDao`, y los métodos necesarios para acceder a ella, `getPartyDao` y `setPartyDao`. Esta propiedad contendrá el DAO responsable de la persistencia de la clase `Party` y sus subclases y será inicializado por Spring a partir del contexto de aplicación.

Cada uno de los métodos de la fachada debe ejecutarse en una transacción, lo que se puede configurar en el contexto de aplicación de Spring, que se puede ver en el Ejemplo 8.11, “Configuración de la fachada y los DAOs de party”.

**Ejemplo 8.11. Configuración de la fachada y los DAOs de party**

```
<!-- ======================== PERSISTENCE DEFINITIONS ======================== -->

<bean id="mappingResources" class="java.util.ArrayList">❶
    <constructor-arg>
        <list>
            <value>net/sf/oness/party/model/party/bo/Party.hbm.xml</value>
        </list>
    </constructor-arg>
</bean>

<!-- DAO objects: Hibernate implementation -->
<bean id="partyDao"
      class="net.sf.oness.common.model.dao.hibernate.HibernateDao">❷
    <constructor-arg>
        <value>net.sf.oness.party.model.party.bo.Party</value>
    </constructor-arg>
    <property name="sessionFactory">
        <ref bean="sessionFactory" />
    </property>
</bean>

<!-- ================================ FACADE ================================ -->

<!-- Party Facade -->
<bean id="partyFacadeDelegate"
    class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">❸
    <property name="transactionManager">
        <ref bean="transactionManager" />
    </property>
```

```
    <property name="target">
        <ref local="partyTarget" />
    </property>
    <property name="transactionAttributes">
        <props>
            <prop key="get*">readOnly</prop>
            <prop key="find*">readOnly</prop>
            <prop key="edit*">readOnly</prop>
            <prop key="create*">PROPAGATION_REQUIRED,ISOLATION_SERIALIZABLE</prop>
            <prop key="update*">PROPAGATION_REQUIRED,ISOLATION_SERIALIZABLE</prop>
            <prop key="delete*">PROPAGATION_REQUIRED,ISOLATION_SERIALIZABLE</prop>
        </props>
    </property>
</bean>

<!-- PartyTarget primary business object implementation -->
<bean id="partyTarget"
      class="net.sf.oness.party.model.facade.PartySpringFacadeDelegate">❹
    <property name="partyDao">
        <ref local="partyDao" />
    </property>
</bean>
```

❶ Lista de ficheros de configuración de Hibernate especificando los detalles de persistencia de las distintas clases

❷ DAO para la clase `Party`

❸ Proxy de la fachada que proporciona las características transaccionales, delegando en la fachada definida en la propiedad `target`

❹ Fachada en la que serán delegadas las llamadas al Proxy

#### 8.1.4.2. Aplicación web

La aplicación web actuará como interfaz del modelo anteriormente creado. Se divide en dos partes: controlador y vista, que serán implementadas utilizando Struts, Tiles, JSP, JSTL y CSS.

Se incluye un fichero de configuración para Tomcat, ya listado anteriormente en el Ejemplo 8.6, “Configuración de la aplicación web party-webapp en Tomcat utilizando JNDI”, que permite configurar la aplicación vía JNDI sin necesidad de modificar el fichero war.

## 8.1.4.2.1. Controlador

El controlador constará de la clase `PartyAction`, que gestionará todas aquellas acciones que realice el usuario a través de la vista y las transformará en llamadas a la fachada de la capa modelo.

- `create`: crear un contacto
- `update`: actualizar un contacto

- `find`: buscar un contacto por sus atributos
- `show`: mostrar un contacto concreto
- `edit`: mostrar un contacto para su posterior posible actualización
- `delete`: borrar un contacto

#### 8.1.4.2.2. Vista

La vista estará formada por las páginas JSP, los mensajes de la aplicación y los ficheros de configuración.

- Interfaz web

  Utilizando Tiles es posible utilizar una aproximación basada en componentes para realizar el interfaz web, separando las páginas JSP en porciones reutilizables que se ensamblarán en los ficheros de configuración de Tiles mediante definiciones.

  - Menús de la aplicación
  - Páginas JSP necesarias para proporcionar la funcionalidad de gestión de contactos.
    - `details.jsp`: muestra los detalles de un contacto.
    - `form.jsp`: que se utiliza tanto para crear o editar un contacto como para buscar por sus atributos.
    - `list.jsp`: muestra una lista de contactos, resultado de una búsqueda.
  - Páginas de soporte: cabecera, pie de página, menús,...
- Mensajes

  Para permitir la internacionalización de la aplicación se utiliza el soporte que proporciona JSTL. A cada mensaje se le asigna una clave que es la que se utiliza en las páginas JSP y en ficheros de texto, uno por idioma, se escriben las correspondencias entre clave y mensaje.
- Configuración
  - Descriptor de aplicaciones web
    - Ubicación de los mensajes para JSTL
    - Ubicación de los ficheros de configuración de Spring

- Acciones y formularios de struts
- Validación de los formularios de struts
- Definiciones de tiles
- Menús de struts-menu

## 8.2. Segunda iteración

En esta segunda iteración se completará la gestión de contactos y se añadirá la funcionalidad de autenticación y autorización, como se puede ver en la Tabla 8.2, “Historias segunda iteración”.

**Tabla 8.2. Historias segunda iteración**

| | |
|---|---|
| Añadir información de contacto a un contacto | 1 |
| Visualización, modificación y eliminación de información de contacto | 1 |
| Autenticación y autorización | 2 |
| **ESTIMACIÓN INICIAL** | **4** |
| **REAL** | **2** |

### 8.2.1. Añadir información de contacto a un contacto

En primer lugar se diseñan los objetos del dominio que modelan la información de contacto, dirección postal, número de teléfono, dirección de correo electrónico y dirección web, cuyo diagrama UML se puede ver en la Figura 8.2, “Información de contacto”.

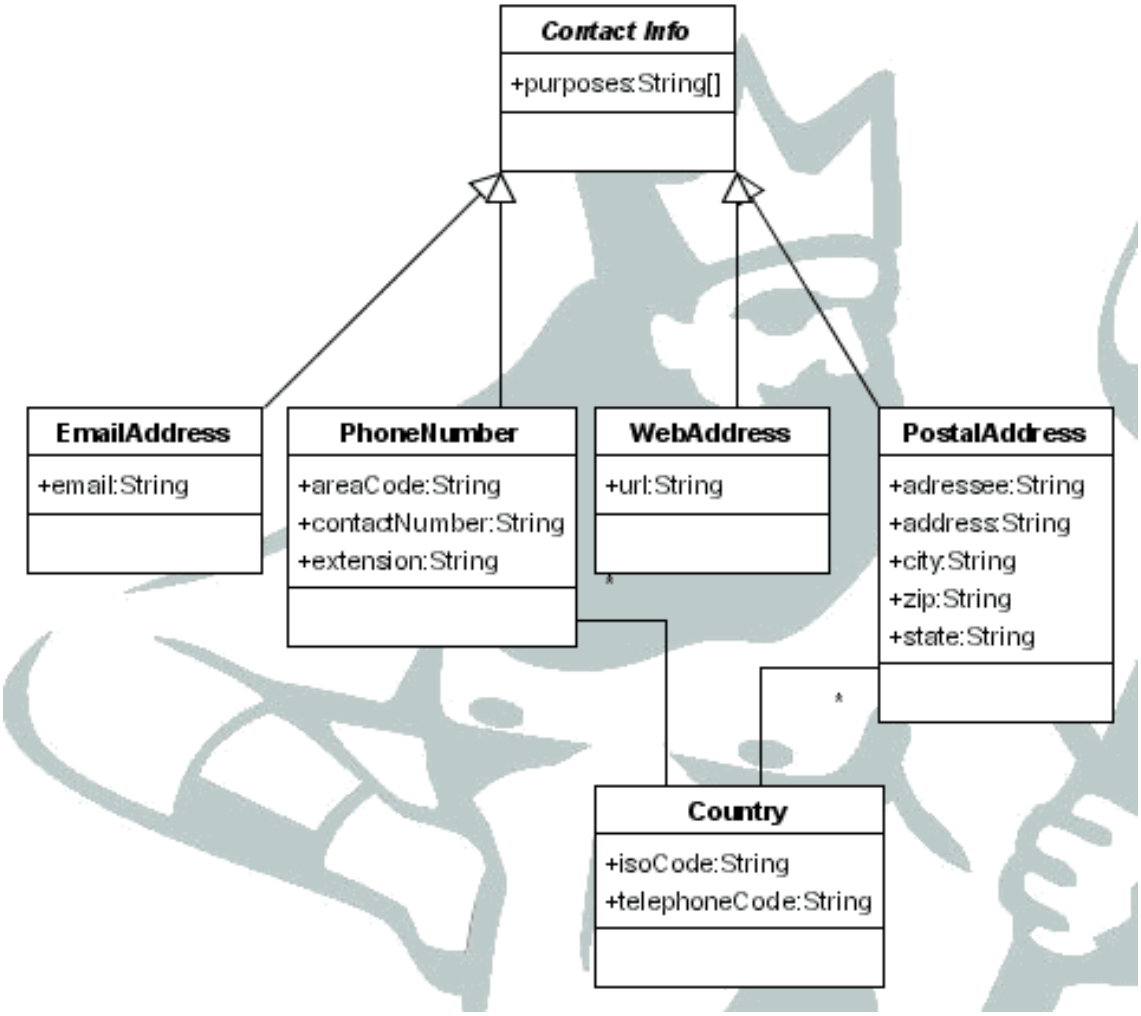


**Figura 8.2. Información de contacto**

Para exponer esta nueva funcionalidad a las capas superiores se añadirá a la fachada el método necesario para crear esta información para un contacto, quedando con se ve en la Figura 8.3, “Añadir información de contacto a un contacto”.

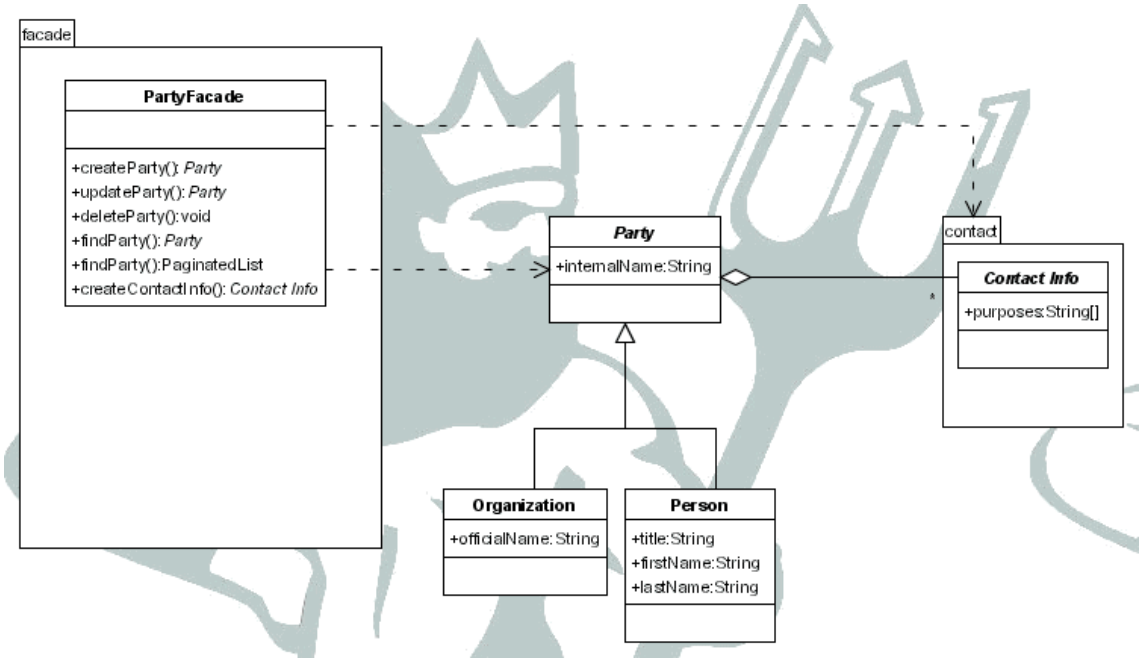


**Figura 8.3. Añadir información de contacto a un contacto**

En cuanto a las capas controlador y vista, en la primera se añadirá una acción para gestionar la información de contacto, en principio con el método necesario para crear la información de contacto (`create`), junto con la configuración necesaria, mientras que en la segunda se añadirán más mensajes de aplicación, así como dos páginas jsp, una con el formulario necesario para introducir los datos en el momento de la creación y otra para mostrar una lista de informaciones de contacto, modificando la página que mostraba los detalles de contacto para incluir la opción de añadir información de contacto así como la lista referente al contacto que se está visualizando.

En los casos de número de teléfono y dirección postal, en los que es necesario especificar el país al que se refieren, es necesario presentar una lista de países para que el usuario pueda seleccionar uno de ellos. La mejor forma de implementar esta funcionalidad es mediante una librería de tags que pueda ser reutilizada, lo que implica que debería estar dentro del módulo *common-webapp*. Esto, y el hecho de que posteriormente será necesario factorizar los recursos comunes de las aplicaciones web hace que el módulo *common-webapp* pase a ser *common-webapp-controller*, y se creará un nuevo módulo *common-webapp-taglib* donde estará contenida la librería de tags para mostrar los países.

#### 8.2.1.1. Librería de tags de países internacionalizable

Esta librería debe proporcionar dos funciones:

- mostrar una lista de países en un formulario, en forma de lista desplegable donde el usuario pueda seleccionar uno de ellos.
- mostrar el nombre de un país en el idioma del usuario.

Los países se identificarán por su código ISO y en principio se añadirán los nombres en inglés, español y francés.

Para facilitar su uso se implementa soporte para el lenguaje de expresiones de JSTL basado en la implementación de referencia de JSTL realizada por el proyecto *Jakarta taglibs* dentro de la *Apache Software Foundation*.

## 8.2.2. Visualización, modificación y eliminación de información de contacto

Con este fin se añadirán los métodos `updateContactInfo`, `deleteContactInfo` y `findContactInfo` a la fachada del modelo, tal y como se muestra en la Figura 8.4, “Visualización, modificación y eliminación de información de contacto”.

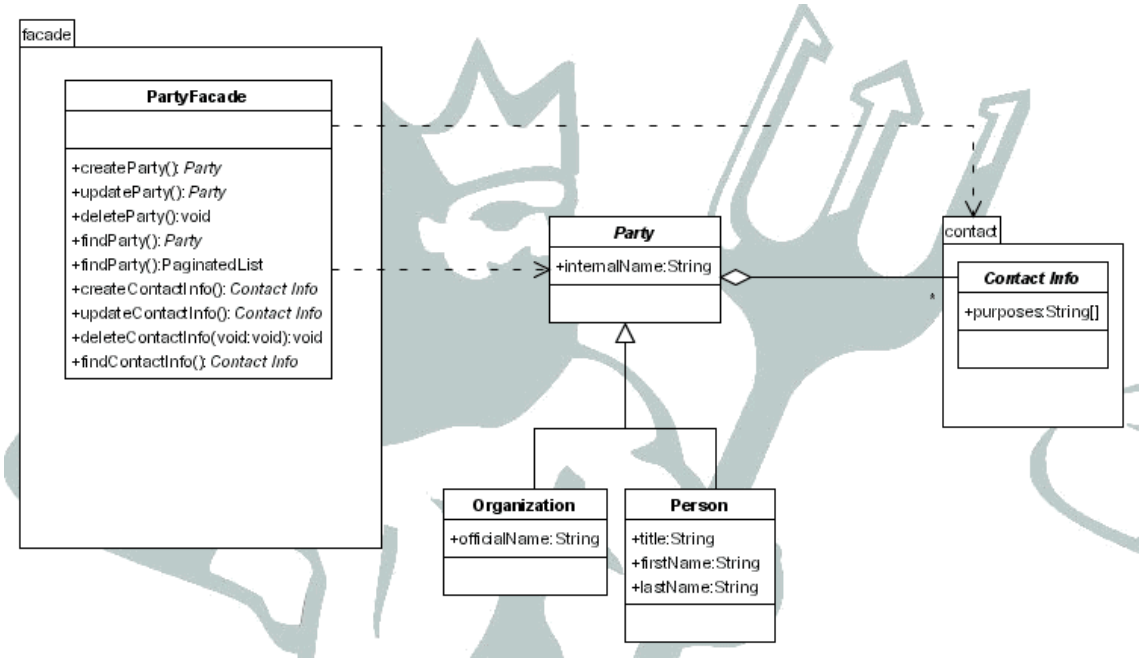


**Figura 8.4. Visualización, modificación y eliminación de información de contacto**

En el controlador será necesario añadir los correspondientes métodos a la acción `ContactInfoAction` anteriormente implementada: `update`, `show`, `edit` y `delete`, para actulizar, mostrar, mostrar para editar y borrar respectivamente, junto con la configuración necesaria de Struts.

En la vista tan sólo será necesario añadir la página jsp para mostrar los detalles de una información de contacto, sea del tipo que sea, y la configuración de las definiciones de Tiles correspondientes.

## 8.2.3. Autenticación y autorización

Para gestionar la autenticación y autorización se utilizará *Acegi Security System for Spring*, que permite integrar de manera sencilla y no intrusiva toda una serie de características de seguridad en una aplicación que utilice Spring.

### 8.2.3.1. Modelo

Acegi proporciona implementaciones con las que la información de usuario puede estar en memoria, útil en entornos de prueba, en una base de datos accesible vía JBDC, o en un servicio JAAS. Para obtener una total independencia del sistema gestor de base de datos será necesario realizar una implementación que utilice Hibernate para acceder a la información. Para ello tan sólo será necesario crear las clases `User` y `Authority`, que representan respectivamente usuarios, con nombre, contraseña y estado (habilitado o deshabilitado), y los roles o grupos a los que pertenece.

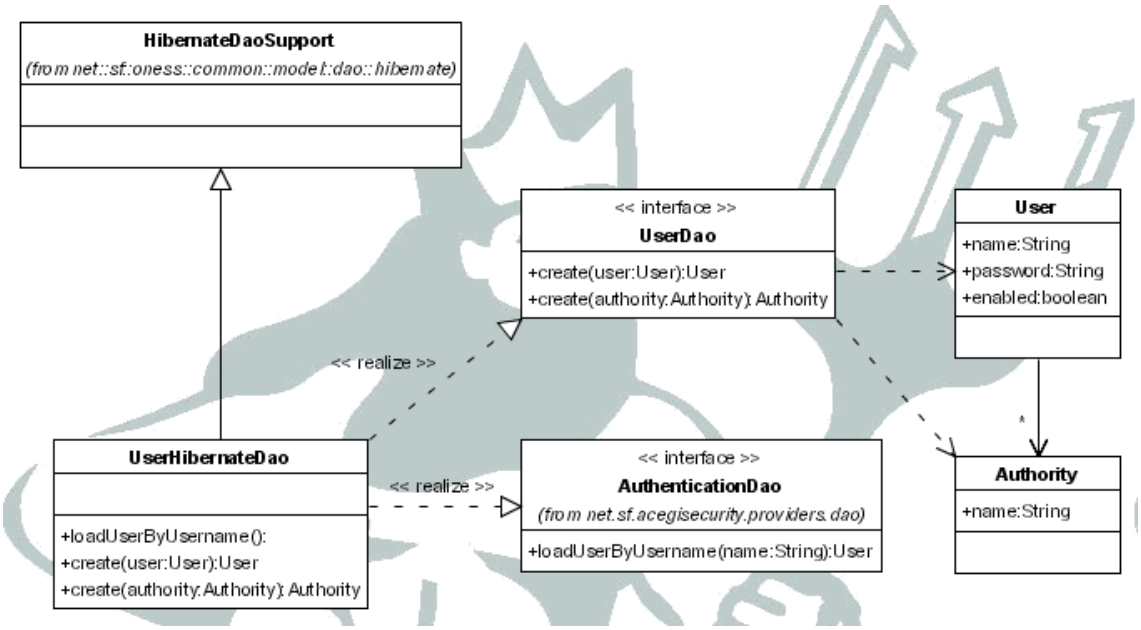


**Figura 8.5. Gestión de usuarios**

En cuanto a la configuración de Acegi ésta se realiza dentro del contexto de aplicación de Spring, en un fichero `applicationContext-oness-user-model.xml` que puede ser incluido o excluido en la configuración, habilitando o no las características de autenticación y

autorización, útil para la realización de tests.

**Ejemplo 8.12. Configuración de autenticación y autorización en el modelo**

```
<beans>

    <!-- User Dao -->
    <bean id="userDao" class="net.sf.oness.user.model.dao.UserHibernateDao"> ❶
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>

    <bean id="mappingResources-user" class="java.util.ArrayList"> ❷
        <constructor-arg>
            <list>
                <value>net/sf/oness/user/model/bo/Authority.hbm.xml</value>
                <value>net/sf/oness/user/model/bo/User.hbm.xml</value>
            </list>
        </constructor-arg>
    </bean>

    <!-- ==================== AUTHENTICATION DEFINITIONS ================== -->

    <!-- Data access object which stores authentication information -->
    <!-- Hibernate implementation -->
    <bean id="authenticationDao"
          class="net.sf.oness.user.model.dao.UserHibernateDao"> ❸
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>

    <!-- ============ SECURITY BEANS YOU WILL RARELY (IF EVER) CHANGE ========== -->

    <bean id="daoAuthenticationProvider"
          class="net.sf.acegisecurity.providers.dao.DaoAuthenticationProvider"> ❹
        <property name="authenticationDao"><ref local="authenticationDao"/></property>
        <property name="userCache"><ref local="userCache"/></property>
        <!--
        <property name="saltSource"><ref local="saltSource"/></property>
        <property name="passwordEncoder"><ref local="passwordEncoder"/></property>
        -->
    </bean>

    <bean id="passwordEncoder"
          class="net.sf.acegisecurity.providers.encoding.Md5PasswordEncoder"/> ❺

    <bean id="userCache"
          class="net.sf.acegisecurity.providers.dao.cache.EhCacheBasedUserCache"> ❻
        <property name="minutesToIdle"><value>5</value></property>
    </bean>

    <bean id="authenticationManager"
          class="net.sf.acegisecurity.providers.ProviderManager"> ❼
        <property name="providers">
          <list>
            <ref local="daoAuthenticationProvider"/>
          </list>
```

```
        </property>
    </bean>

    <bean id="roleVoter" class="net.sf.acegisecurity.vote.RoleVoter"/> ❽

    <bean id="accessDecisionManager"
          class="net.sf.acegisecurity.vote.AffirmativeBased"> ❾
        <property name="allowIfAllAbstainDecisions"><value>false</value></property>
        <property name="decisionVoters">
          <list>
            <ref local="roleVoter"/>
          </list>
        </property>
    </bean>

</beans>
```

❶ Definición del DAO que gestiona la persistencia de usuarios independientemente de Acegi
❷ Ficheros de configuración de Hibernate correspondientes a las clases `User` y `Authority`
❸ Definición del DAO que gestiona la persistencia de usuarios dentro de Acegi
❹ Proveedor de autenticación que utiliza un DAO para acceder a la información, donde se puede configurar entre otros el cifrado de contraseñas y la caché de usuarios
❺ Cifrador de contraseñas utilizando MD5
❻ Caché de usuarios y contraseñas utilizando EHCache
❼ Gestor de proveedores donde se indica la utilización del proveedo anteriormente definido
❽ Toma de decisiones basada en roles
❾ Gestor de decisiones de acceso basado en votos afirmativos

Será necesario añadir los ficheros de mapeado de Hibernate correspondientes a la gestión de usuarios a los correspondientes a cada módulo, para lo que se crea una clase utilidad `ListConcatenator` que se puede utilizar como se muestra en el Ejemplo 8.13, “Concatenación de listas en Spring”.

**Ejemplo 8.13. Concatenación de listas en Spring**

```
    <bean id="mappingResources"
          class="net.sf.oness.common.model.util.spring.ListConcatenator">
        <constructor-arg>
            <list>
                <ref local="mappingResources-party"/>
                <ref bean="mappingResources-user"/>
            </list>
        </constructor-arg>
    </bean>

    <bean id="mappingResources-party" class="java.util.ArrayList">
        <constructor-arg>
            <list>
                <value>net/sf/oness/party/model/party/bo/Party.hbm.xml</value>
                <value>net/sf/oness/party/model/contact/bo/ContactInfo.hbm.xml</value>
```

```
                <value>net/sf/oness/party/model/contact/bo/Country.hbm.xml</value>
            </list>
        </constructor-arg>
    </bean>
```

En este momento se pueden completar las tareas pendientes de la primera iteración en cuanto a la auditoría se refiere. Para ello se crea la clase `SecurityHelper` con el método `getUserName` dentro del módulo *common-model* que delegará en el `ContextHolder` proporcionado por Acegi. Así se podrá acceder de forma sencilla al nombre de usuario desde `AuditingDaoHelper` para guardarlo como responsable de los cambios realizados.

#### 8.2.3.2. Aplicación web

Para añadir autenticación y autorización a las aplicaciones web se utilizará tambien el soporte que ofrece Acegi.

Acegi Security permite que la aplicación se pueda integrar de manera sencilla en el servicio central de autenticación CAS desarrollado por la universidad de Yale, teniendo así funcionalidad de *Single Sign On*, lo que permite que distintas aplicaciones se autentiquen en un único punto y evitando la necesidad de que los usuarios introduzcan su nombre y contraseña en cada una de ellas. La utilización o no de CAS es cuestión de configuración, pudiendo habilitarlo y deshabilitarlo sin que afecte al funcionamiento del sistema.

El módulo *user* se desarrollará para que pueda funcionar tanto con CAS como sin él, que será el funcionamiento por defecto ya que la instalación de un servidor CAS aún siendo una simple aplicación web requiere la creación de un certificado SSL que debe ser añadido en el directorio donde reside la máquina virtual Java así como la configuración del servidor de aplicaciones para activar el soporte HTTPS, siendo demasiados requisitos como para que la aplicación sea fácilmente probada por aquellas personas que se la descarguen de el sitio web en [Sourceforge]. Para más información sobre la configuración de la integración con CAS se puede consultar la documentación de Acegi.

El sistema utilizado por Acegi para implementar los mecanismos de seguridad en los recursos web está basado en el uso de filtros, que interceptan las peticiones HTTP y realizan algún tipo de procesamiento tomando las medidas oportunas.

**Ejemplo 8.14. Configuración de autenticación y autorización en el descriptor de aplicación web**

```
<!-- cas -->
<context-param>
  <param-name>edu.yale.its.tp.cas.authHandler</param-name>❶
  <param-value>net.sf.acegisecurity.adapters.cas.CasPasswordHandlerProxy</param-value>
```

```
</context-param>

<!-- Required for CAS ProxyTicketReceptor servlet. This is the
     URL to CAS' "proxy" actuator, where a PGT and TargetService can
     be presented to obtain a new proxy ticket. THIS CAN BE
     REMOVED IF THE APPLICATION DOESN'T NEED TO ACT AS A PROXY -->
<context-param>
  <param-name>edu.yale.its.tp.cas.proxyUrl</param-name>❷
  <param-value>http://localhost:8433/cas/proxy</param-value>
</context-param>

<!-- Acegi Security Filters -->
<!--
<filter>❸
  <filter-name>Acegi Channel Processing Filter</filter-name>
  <filter-class>net.sf.acegisecurity.util.FilterToBeanProxy</filter-class>
  <init-param>
    <param-name>targetClass</param-name>
    <param-value>
      net.sf.acegisecurity.securechannel.ChannelProcessingFilter
    </param-value>
  </init-param>
</filter>
-->
<filter>❹
  <filter-name>Acegi Authentication Processing Filter</filter-name>
  <filter-class>net.sf.acegisecurity.util.FilterToBeanProxy</filter-class>
  <init-param>
    <param-name>targetClass</param-name>
    <!-- Not using CAS -->
    <param-value>
      net.sf.acegisecurity.ui.webapp.AuthenticationProcessingFilter
    </param-value>
    <!-- Using CAS -->
    <!--
    <param-value>
      net.sf.acegisecurity.ui.cas.CasProcessingFilter
    </param-value>
    -->
  </init-param>
</filter>
<!--
<filter>❺
  <filter-name>Acegi CAS Processing Filter</filter-name>
  <filter-class>net.sf.acegisecurity.util.FilterToBeanProxy</filter-class>
  <init-param>
    <param-name>targetClass</param-name>
    <param-value>net.sf.acegisecurity.ui.cas.CasProcessingFilter</param-value>
  </init-param>
</filter>
<filter>❻
  <filter-name>Acegi HTTP BASIC Authorization Filter</filter-name>
  <filter-class>net.sf.acegisecurity.util.FilterToBeanProxy</filter-class>
  <init-param>
    <param-name>targetClass</param-name>
    <param-value>net.sf.acegisecurity.ui.basicauth.BasicProcessingFilter</param-value>
  </init-param>
</filter>
-->
<filter>❼
  <filter-name>Acegi Security System for Spring Auto Integration Filter</filter-name>
```

```
  <filter-class>net.sf.acegisecurity.ui.AutoIntegrationFilter</filter-class>
</filter>
<filter>❽
  <filter-name>Acegi HTTP Request Security Filter</filter-name>
  <filter-class>net.sf.acegisecurity.util.FilterToBeanProxy</filter-class>
  <init-param>
    <param-name>targetClass</param-name>
    <param-value>
      net.sf.acegisecurity.intercept.web.SecurityEnforcementFilter
    </param-value>
  </init-param>
</filter>

❾
<!--
<filter-mapping>
  <filter-name>Acegi Channel Processing Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
-->
<filter-mapping>
  <filter-name>Acegi Authentication Processing Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<!--
<filter-mapping>
  <filter-name>Acegi CAS Processing Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>Acegi HTTP BASIC Authorization Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
-->
<filter-mapping>
  <filter-name>Acegi Security System for Spring Auto Integration Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
  <filter-name>Acegi HTTP Request Security Filter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<!-- CAS servlet which receives a proxy-granting ticket from the CAS
     server. THIS CAN BE REMOVED IF THE APPLICATION DOESN'T NEED TO
     ACT AS A PROXY -->
<!--
<servlet>❿
  <servlet-name>casproxy</servlet-name>
  <servlet-class>edu.yale.its.tp.cas.proxy.ProxyTicketReceptor</servlet-class>
  <load-on-startup>3</load-on-startup>
</servlet>
-->

<!-- CAS Proxy -->
<!--
<servlet-mapping>
  <servlet-name>casproxy</servlet-name>
  <url-pattern>/casProxy/*</url-pattern>
</servlet-mapping>
-->
```

❶ Parámetro necesario en caso de utilizar CAS
❷ URL donde está accesible el servidor CAS
❸ Filtro que permite definir la utilización de HTTPS para ciertas urls
❹ Filtro que proporciona la autenticación a través de un formulario en una pagina web (HTTP Session), bien en la propia aplicación web o en un servidor CAS
❺ Filtro que construirá las peticiones al servidor CAS
❻ Filtro que utiliza autenticación mediante HTTP Basic, que preguntará al usuario nombre y contraseña en una diálogo del navegador en en lugar de un formulario en una página web.
❼ Filtro que determina automáticamente el filtro a utilizar, utilice HTTP Session, HTTP Basic o los mecanismos proporcionados por el contenedor
❽ Filtro que permite controlar el acceso a determinadas urls
❾ URLs a las que se aplicará cada filtro, normalmente se aplicarán a todas
❿ Servlet que permitiría que esta aplicación web funcionara como servidor CAS

La configuración de los distintos filtros en el contexto de aplicación de Spring se puede ver en el Ejemplo 8.15, “Configuración de autenticación y autorización en la aplicación web”.

**Ejemplo 8.15. Configuración de autenticación y autorización en la aplicación web**

```
<bean id="authenticationProcessingFilter"
      class="net.sf.acegisecurity.ui.webapp.AuthenticationProcessingFilter"> ❶
    <property name="authenticationManager">
        <ref bean="authenticationManager"/>
    </property>
    <property name="authenticationFailureUrl">
        <value><![CDATA[/show.do?page=.login&login_error=1]]></value>
    </property>
    <property name="defaultTargetUrl"><value>/</value></property>
    <property name="filterProcessesUrl"><value>/security_check</value></property>
</bean>

<bean id="securityEnforcementFilter"
      class="net.sf.acegisecurity.intercept.web.SecurityEnforcementFilter"> ❷
    <property name="filterSecurityInterceptor">
        <ref bean="filterInvocationInterceptor"/>
    </property>
    <property name="authenticationEntryPoint">
        <ref local="authenticationProcessingFilterEntryPoint"/>
    </property>
</bean>

<bean id="authenticationProcessingFilterEntryPoint"
    class="net.sf.acegisecurity.ui.webapp.AuthenticationProcessingFilterEntryPoint"> ❸
    <property name="loginFormUrl">
        <value><![CDATA[/show.do?page=.login]]></value>
    </property>
    <property name="forceHttps"><value>false</value></property>
</bean>

<!-- Use basic authentication -->
<!--
<bean id="basicProcessingFilter"
```

```
      class="net.sf.acegisecurity.ui.basicauth.BasicProcessingFilter"> ❹
    <property name="authenticationManager">
        <ref bean="authenticationManager"/>
    </property>
    <property name="authenticationEntryPoint">
        <ref bean="basicProcessingFilterEntryPoint"/>
    </property>
</bean>

<bean id="basicProcessingFilterEntryPoint"
      class="net.sf.acegisecurity.ui.basicauth.BasicProcessingFilterEntryPoint"> ❺
    <property name="realmName"><value>ONess Realm</value></property>
</bean>
-->
```

❶ Configuración del filtro de autenticación, especificando la url en caso de que el login sea incorrecto

❷ Configuración donde se define el tipo de punto de entrada según se utilice autenticación mediante http basic o formulario

❸ Configuración de la url donde se encuentra el formulario de login y si se debe usar https

❹ Configuración necesaria en caso de utilizar http basic

❺ Nombre del dominio en caso de utilizar http basic

Un ejemplo de configuración de reglas se puede ver en el Ejemplo 8.16, “Configuración de reglas de autorización”. En él se configura el filtro que permite o restringe el acceso a distintas urls según el grupo al que pertenece el usuario. En este caso se requiere que todas aquellas operaciones que impliquen una modificación de datos sólo puedan ser realizadas por usuarios no anónimos. Los usuarios anónimos podrán acceder a otras urls como búsqueda y consulta. Un usuario dentro del grupo administrador podrá acceder a todas las urls y será el único que podra hacerlo a aquellas que se encuentren bajo /secure/.

**Ejemplo 8.16. Configuración de reglas de autorización**

```
<bean id="filterInvocationInterceptor"
      class="net.sf.acegisecurity.intercept.web.FilterSecurityInterceptor">
    <property name="authenticationManager">
        <ref bean="authenticationManager"/>
    </property>
    <property name="accessDecisionManager">
        <ref bean="accessDecisionManager"/>
    </property>
    <property name="objectDefinitionSource">
        <value>
            CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
            PATTERN_TYPE_APACHE_ANT
            /secure/**=ROLE_ADMIN
            /**/*create*=ROLE_USER,ROLE_ADMIN
            /**/*edit*=ROLE_USER,ROLE_ADMIN
            /**/*update*=ROLE_USER,ROLE_ADMIN
            /**/*delete*=ROLE_USER,ROLE_ADMIN
```

```
        </value>
    </property>
</bean>
```

Dado que gran parte de los recursos anteriormente desarrollados en el módulo *party* serán compartidos por el módulo user y otras futuras aplicaciones web, surge la necesidad de separarlos para poder ser reutilizados. La situación ideal sería crear otra aplicación web con los recursos comunes y acceder a ellos desde las otras aplicaciones, pero esta aproximación tiene dos inconvenientes:

- no todos los contenedores permiten que una aplicación acceda a los recursos de otra ya que no es una funcionalidad estándar, aunque los más extendidos sí, como Tomcat.
- Tiles no permite utilizar componentes alojados en otro contexto (aplicación web).

Estos inconvenientes, principalmente el segundo, hacen que sea inviable esta solución, con lo cual será necesario crear aplicaciones web que contengan los recursos comunes. Para ello se descompondrá el módulo *common-webapp* en *common-webapp-controller*, que contendrá todo lo que estaba anteriormente en *common-webapp*, y *common-webapp-view*, que contendrá los recursos comunes anteriormente mencionados, y que serán incluidos automáticamente en las aplicaciones web en el momento de su construcción.

Más concretamente el módulo *common-webapp-view* contendrá:

- Mensajes comunes a todo el sistema
- Interfaz web
    - Páginas de error
    - Imágenes
    - JavaScript
    - Hojas de estilo CSS
    - Diseño (cabecera, pie de página,...)
- Configuración
    - Configuración común en el fichero descriptor de aplicación web `web.xml`.
    - Configuración común en el fichero de configuración de Struts `struts-config.xml`.

- Definiciones de *tiles* comunes
- Reglas de validación

Para poder combinar la configuración común en lo ficheros xml del descriptor de aplicación web y de struts con la correspondiente a cada módulo ha sido necesaria la creación de dos hojas de transformación xml que a partir de dos ficheros xml con entradas obtiene uno con las entradas de ambos. Esta aproximación es mucho mejor y más elegante que la utilización de XDoclet que obligaría a crear un gran número de entidades xml, con el consiguiente engorro que supone su manejo por además no ser ficheros xml bien formados.

Se hace imprescindible insertar datos de ejemplo en la base de datos ya que en caso contrario los usuarios no podrían acceder a la funcionalidad que se ha configurado para requerir un usuario no anónimo. Para ello en esta iteración se ha optado por realizar una acción de Struts `DatabasePopulatorAction` en el módulo *common-webapp-controller* que utilizará la clase `DatabasePopulator` del módulo *common-model*. [1]

## 8.3. Tercera iteración

En esta tercera iteración se implementará la funcionalidad relativa a la gestión de inventario y la accesibilidad desde dispositivos móviles de todos los módulos.

**Tabla 8.3. Historias tercera iteración**

| | |
|---|---|
| Creación de modelos y productos | 1 |
| Visualización, modificación, eliminación y búsqueda de modelos | 1 |
| Creación y modificación de precios | 1 |
| Accesibilidad desde dispositivos móviles | 1 |
| **ESTIMACIÓN INICIAL** | **4** |
| **REAL** | **4** |

[1] En la siguiente iteración se ha cambiado este método, configurando un *listener* en la aplicación web que inserta los datos de ejemplo cuando ésta es cargada

En esta iteración han surgido problemas en la implementación de la funcionalidad relativa a la auditoría en el núcleo del sistema, dado que han surgido errores en la integración con Hibernate en las relaciones entre objetos. A pesar de estos problemas la duración de la iteración no ha aumentado, gracias a que añadir nueva funcionalidad ha resultado tener un coste en tiempo realmente bajo.

### 8.3.1. Creación de modelos y productos

Para comenzar se realiza el diagrama de clases UML que se puede ver en la Figura 8.6, "Creación de modelos y productos" con los objetos del dominio descritos en la historia de usuario.

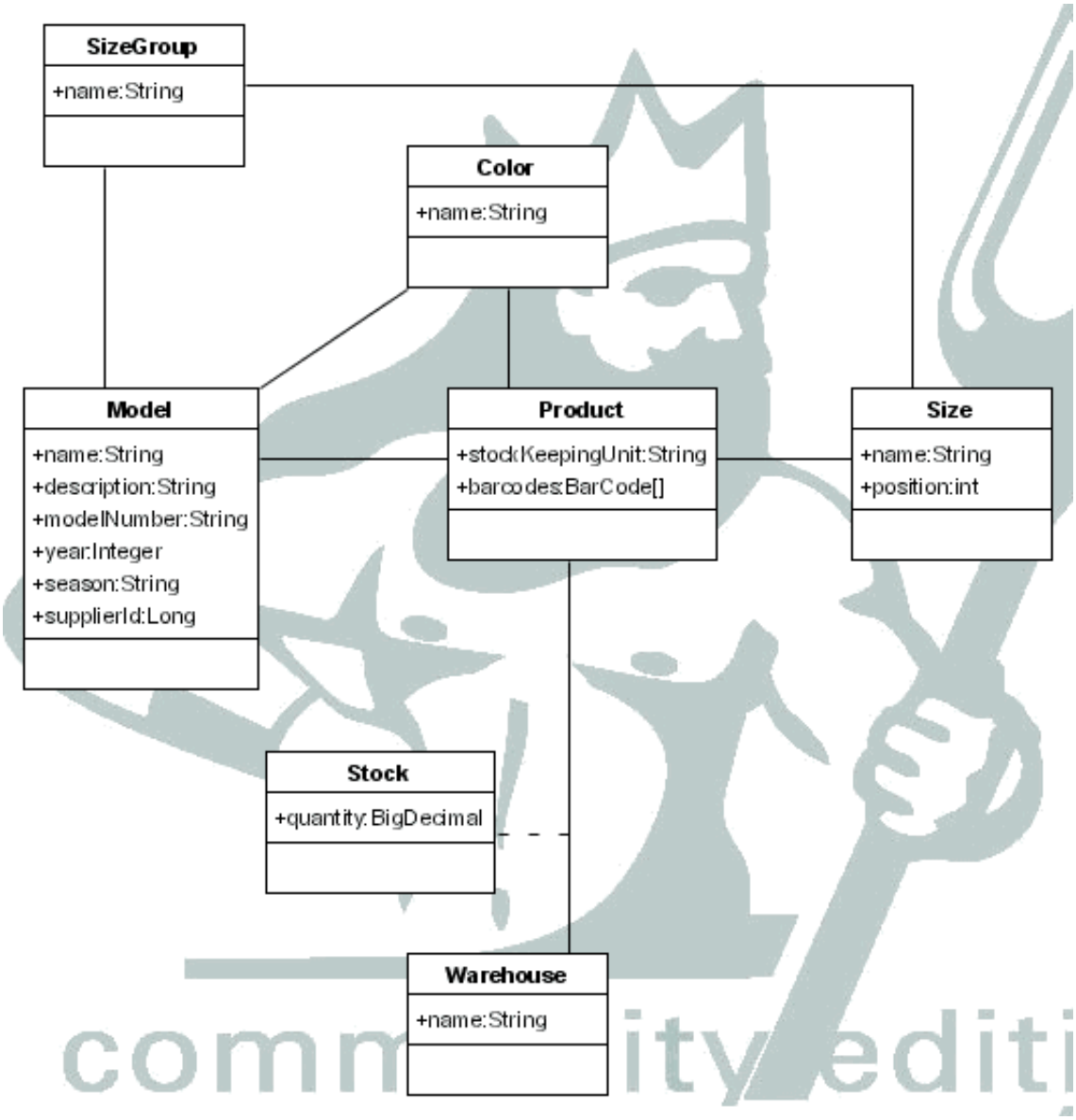


**Figura 8.6. Creación de modelos y productos**

Al igual que en el módulo *party* se creará una fachada *InventoryFacade* del módulo como interfaz de la capa modelo hacia capas superiores ocultando los detalles de implementación de la capa modelo.

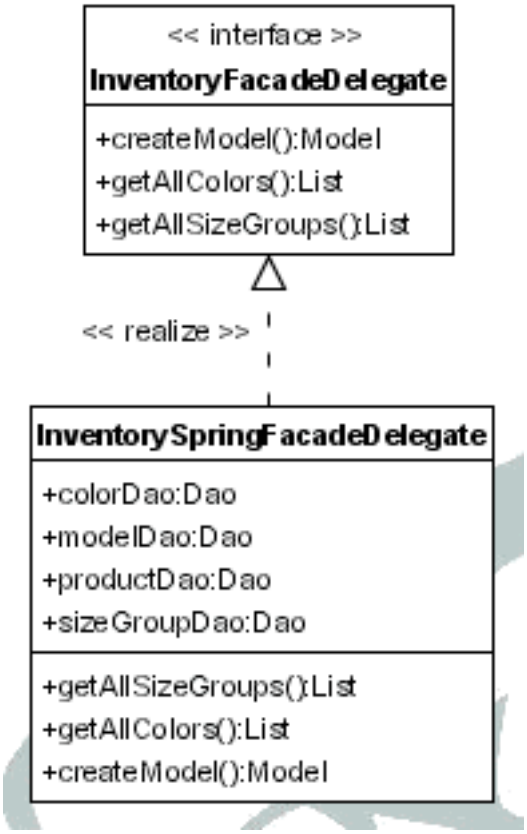


**Figura 8.7. Creación de modelos y productos, fachada**

En la capa controlador se añadirá una acción de Struts, `ModelAction`, que hará de intermediario entre la vista y el modelo, con su correspondiente configuración. En la capa vista al igual que en el módulo *party* se añadirán las páginas jsp para mostrar el formulario de creación y búsqueda de modelos.

Los métodos `getAll*` son necesarios para construír una caché con la información de tallajes y colores en la capa vista sin necesidad de acceder a la base de datos en reiteradas ocasiones. Estos datos no suelen ser modificados por lo que pueden ser cargados al inicio de la aplicación, para lo que se desarrollará un *listener* de la aplicación web, `SpringContextLoaderListener`, que extenderá y sustituirá al proporcionado por Spring, para además de realizar sus operaciones inicializar esta caché. En caso de necesitar reconstruir la caché tan sólo será necesario recargar la aplicación web en el contenedor.

### 8.3.2. Visualización, modificación, eliminación y búsqueda de modelos

Una vez creados los objetos del dominio en la historia anterior en cuanto a la capa modelo sólo será necesario añadir a la fachada los métodos que implementarán esta nueva funcionalidad.

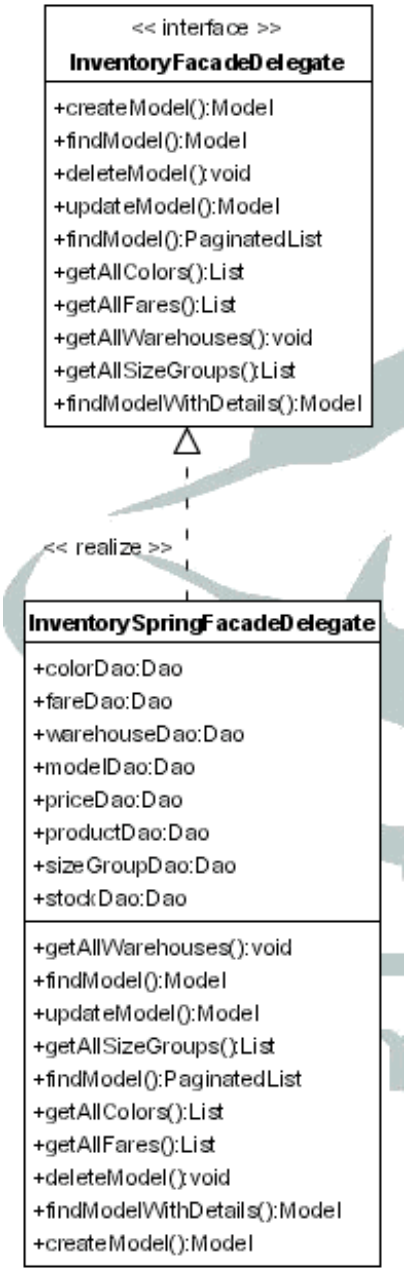


**Figura 8.8. Visualización, modificación, eliminación y búsqueda de modelos, fachada**

En la capa controlador se añadirán a la clase `ModelAction` los métodos que darán soporte a la

nueva funcionalidad (`find`, `edit`, `update`, `delete`) y en la capa vista la página jsp para visualizar un modelo, que mostrará todas las tallas y colores en los que existe, así como sus códigos de barras y las existencias en cada uno de los almacenes.

### 8.3.3. Creación y modificación de precios

Será necesario modelar los precios y tarifas, de forma que cada producto tenga un precio por tarifa, y añadir a la fachada los métodos `editPrices` y `updatePrices`, así como hacer que `findModelWithDetails` devuelva también los precios de cada producto en cada tarifa.

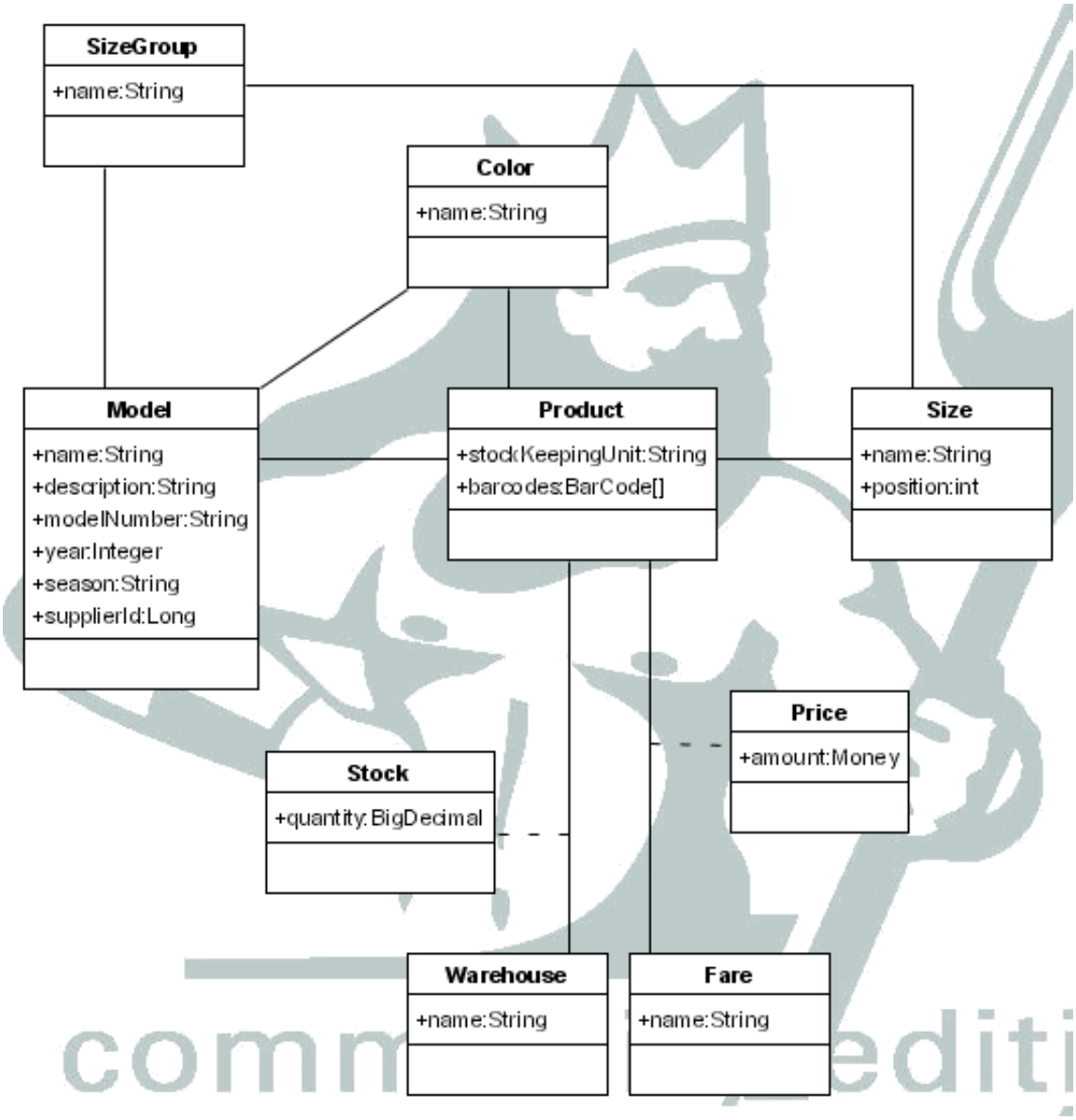


**Figura 8.9. Creación y modificación de precios**

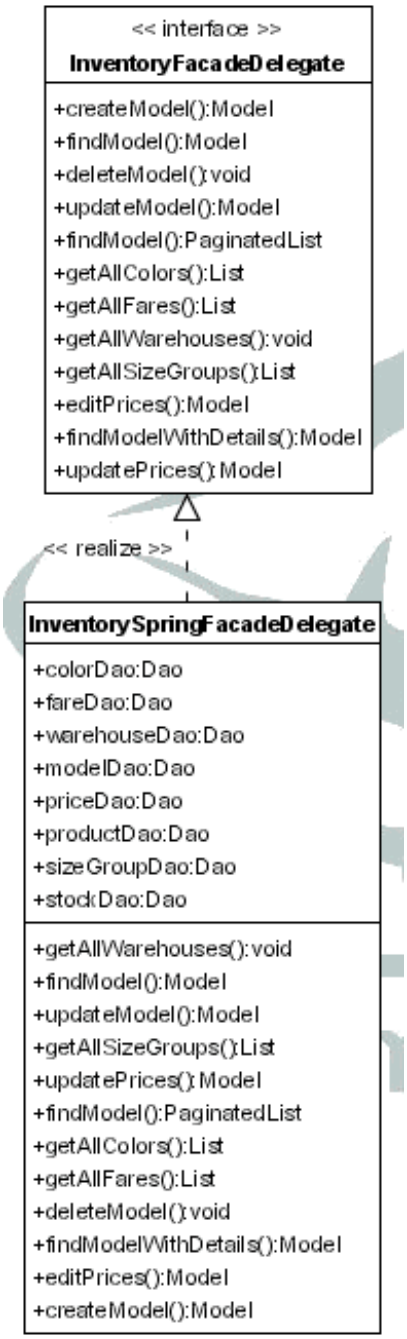


**Figura 8.10. Creación y modificación de precios, fachada**

En el controlador serán necesarias una nueva acción, `PriceAction`, que permitirá visualizar los precios de cada producto de un modelo y su posterior actualización, y las páginas jsp en la vista. Además se añadirá a la visualización de modelos los precios de cada producto en cada una de las tarifas.

### 8.3.4. Accesibilidad desde dispositivos móviles

Dado que la aplicación debe ser accesible desde dispositivos móviles tipo PDA se debe buscar una forma sencilla para adaptar el interfaz web a estos dispositivos, lo que implica principalmente considerar el tamaño de pantalla. Para realizar las pruebas se ha utilizado el simulador del sistema operativo PalmOS junto con su navegador web PalmSource.

La primera opción para adaptar el interfaz ha sido utilizar el soporte que proporciona CSS para estabecer distintas hojas de estilo para distintos fines a través del atributo *media*. Así por ejemplo se ha creado una hoja de estilo para impresión (`media="print"`) que evita que la cabecera, el pie de página o el menú no se impriman, tan sólo el contenido ocupando todo el ancho de la página. Para los dispositivos con pantallas reducidas se puede utilizar el tipo de medio `handheld`, que de igual forma que para el tipo `print` no mostrará ni cabecera ni pie de página, pero que mantendrá el menu en formato texto.

El problema que ha surgido es que el navegador PalmSource no utiliza las hojas de estilo definidas como `handheld`, puede que sea debido a que la calidad de la imagen de la Palm es mucho mejor que la de otros dispositivos portátiles como pueden ser los móviles y los desarrolladores del navegador hayan optado por no utilizar ese tipo de hojas de estilo. Por lo tanto la única opcion restante es utilizar el identificador que el navegador envía con cada petición y cambiar las hojas de estilo si éste se corresponde con el identificador de PalmSource. Dado que las hojas de estilo y otras características de la presentación se definen mediante Tiles en el fichero de configuración de definiciones se ha optado por crear una acción de Tiles, `SwitchLayoutController`, que procesará todas las peticiones y tendrá acceso al contexto de configuración de Tiles, donde se configurará otra disposición de interfaz para PalmSource que sustituirá al interfaz por defecto en caso de que la petición sea de este navegador. Es una solución extensible, pudiéndose añadir tantas definiciones como sea necesario para distintos tipos de navegador o compartir una única definición entre varios.

El interfaz en Internet Explorer en la Figura 8.11, “Interfaz en Internet Explorer, página principal” y la Figura 8.12, “Interfaz en Internet Explorer, resultado de una búsqueda” se puede comparar con el resultado en el simulador de una Palm de la Figura 8.13, “Interfaz en una Palm”, para la página principal y para una página de resultados de búsqueda. Aunque en la página de búsqueda los resultados ocupan más ancho del que ofrece la pantalla se puede desplazar la pantalla tanto de arriba a abajo como de izquierda a derecha utilizando las barras de desplazamiento.

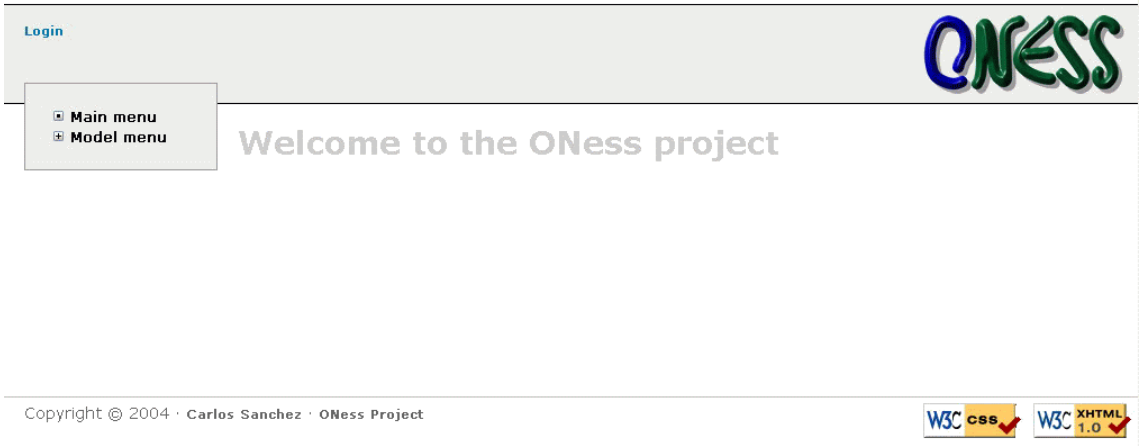


**Figura 8.11. Interfaz en Internet Explorer, página principal**

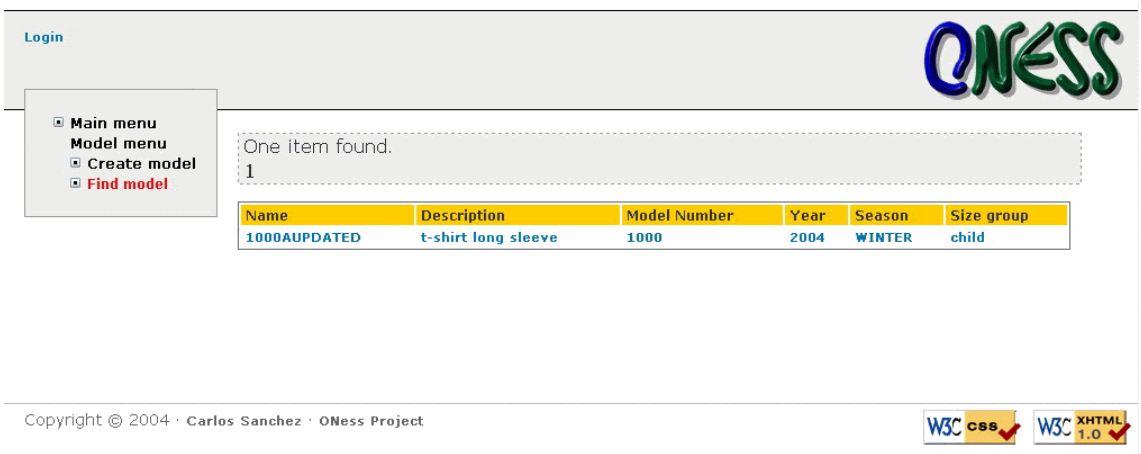


**Figura 8.12. Interfaz en Internet Explorer, resultado de una búsqueda**

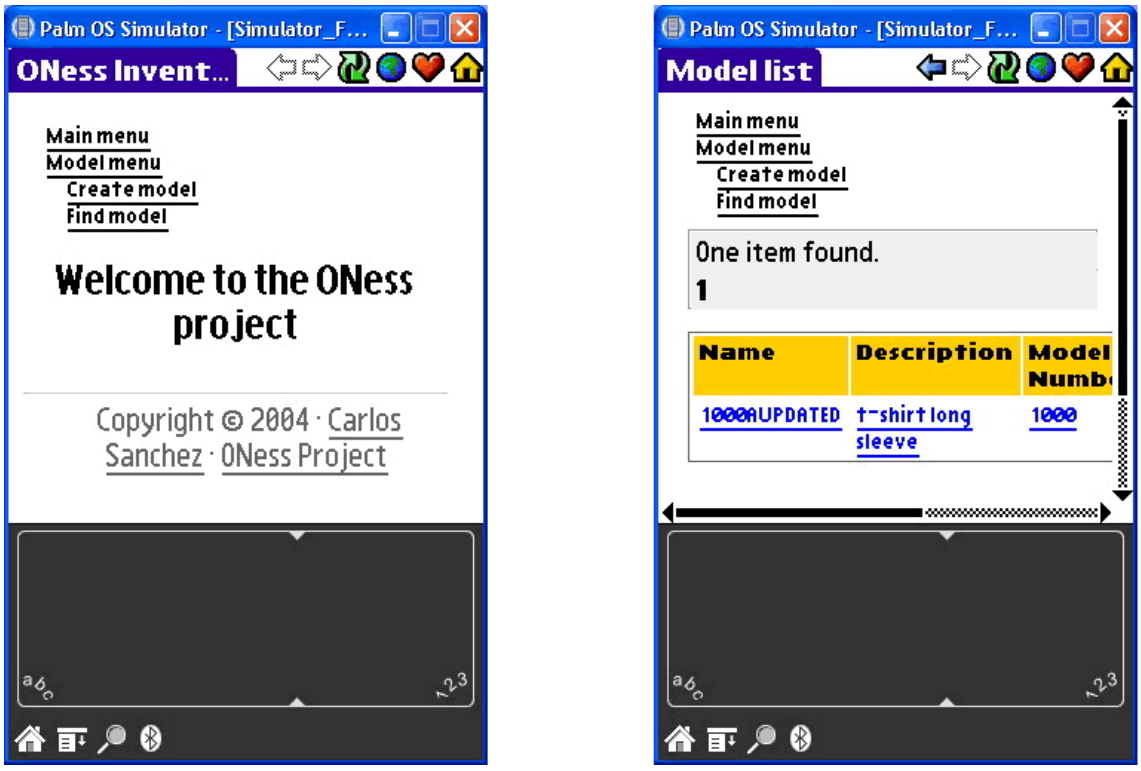


**Figura 8.13. Interfaz en una Palm**

### 8.3.5. Otros cambios

En esta iteración se ha optado por cambiar el método de carga de datos de ejemplo en la base de datos, utilizando para ello el *listener* anteriormente realizado en esta iteración, de forma que los datos son insertados en el momento de cargarse la aplicación web sin necesidad de intervención por parte del usuario.

## 8.4. Cuarta iteración

Esta última iteración concluirá el sistema con la implementación de la funcionalidad relacionada con la gestión de pedidos, albaranes y facturas, tanto para compras como ventas. Como ya se ha comentado anteriormente la introducción de nuevas características es un proceso rápido, cuya duración real es inferior a la inicialmente estimada.

**Tabla 8.4. Historias cuarta iteración**

| | |
|---|---|
| Creación de pedidos y añadir productos a un pedido | 1 |

| | |
|---|---|
| Visualización, modificación, eliminación y listado de pedidos | 1 |
| Creación de albaranes y facturas | 1 |
| **ESTIMACIÓN INICIAL** | **3** |
| **REAL** | **2** |

El proceso de creación de un pedido comenzaría a partir de la visualización de un contacto. Una vez creado se podrán añadir productos desde la visualización de productos, siendo común añadir varios productos a la vez del mismo modelo.

El diagrama UML para los objetos del dominio y la fachada puede verse en la Figura 8.14, “Pedidos, albaranes y facturas”.

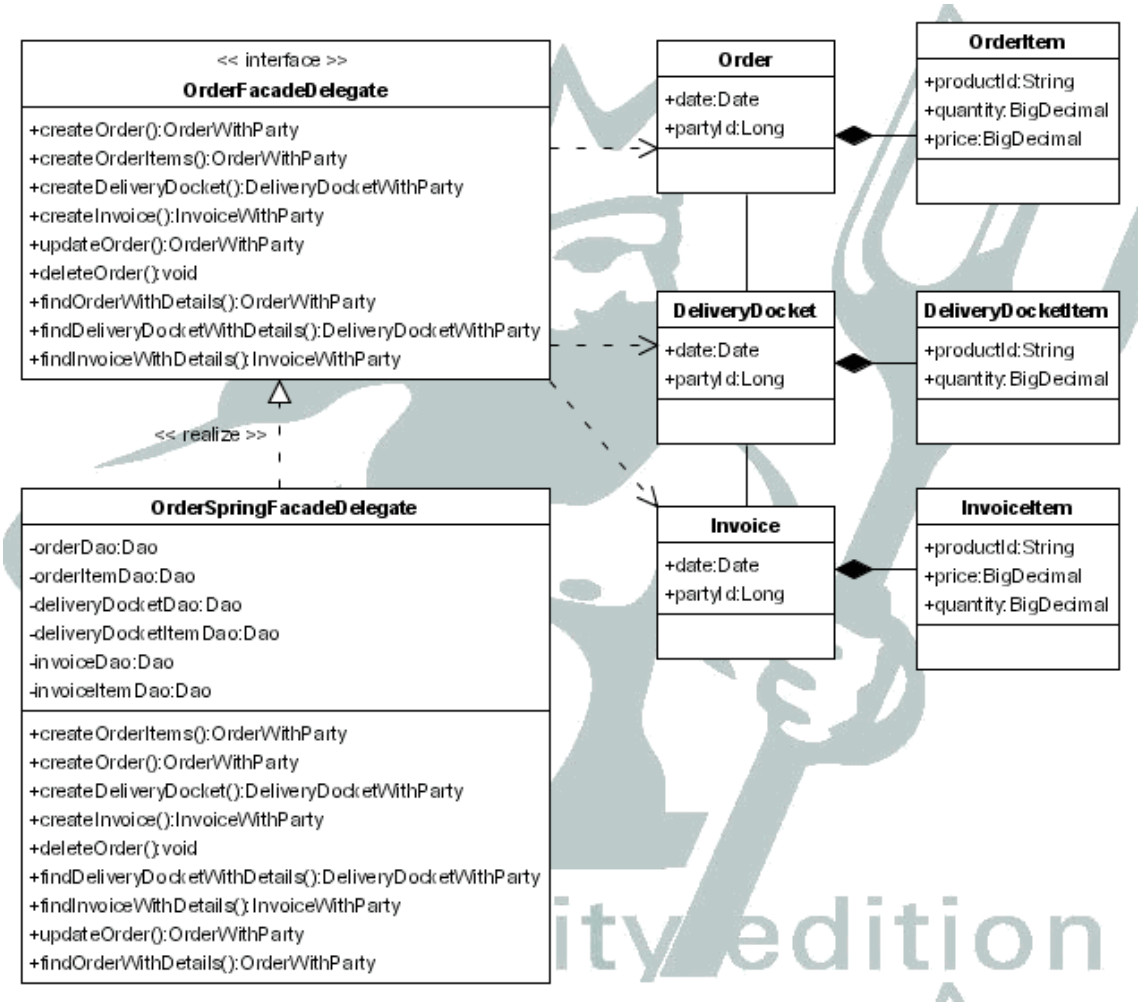


**Figura 8.14. Pedidos, albaranes y facturas**

Para poder mostrar los datos del cliente o proveedor a la vez que se muestran los de pedidos, albaranes o facturas es necesario crear *transfer objects* compuestos, que agrupen ambos objetos. Lo mismo sucede con los productos y las líneas de pedido, albarán o factura. El diagrama UML se puede ver en la Figura 8.15, “Pedidos, albaranes y facturas, transfer objects”.

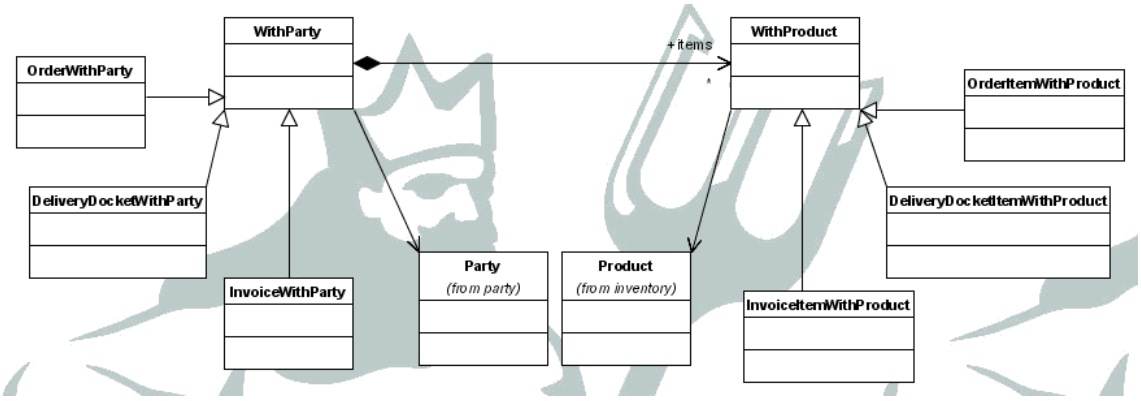


**Figura 8.15. Pedidos, albaranes y facturas, transfer objects**

En cuanto a la vista y controlador será necesario añadir las páginas y acciones para pedidos, albaranes, facturas y las líneas de cada uno de ellos. Además se añadirán las opciones necesarias a visualización de contactos para crear un pedido de ese contacto y visualización de modelos para añadir uno o varios productos al pedido en curso.

### 8.4.1. Otros cambios

Se ha añadido al módulo *inventory* la posibilidad de mostrar una imagen de los modelos para facilitar la interacción con el usuario.

# Capítulo 9. Producción

La fase de producción requiere de pruebas adicionales y revisiones de rendimiento antes de que el sistema sea trasladado al entorno del cliente. Al mismo tiempo, se deben tomar decisiones sobre la inclusión de nuevas características a la versión actual, debido a cambios durante esta fase.

Tras cada iteración se ha puesto a disposición de todo el público a través del sitio web una demostración de lo realizado hasta ese momento, lo que ha permitido comprobar el comportamiento de la aplicación en otro entorno y con usuarios imparciales. Gracias a esta demostración se ha podido detectar algún fallo de integración que ha sido subsanado, así como se facilita la evaluación del proyecto por otros desarrolladores en todo el mundo.

# Parte IV. Conclusiones y trabajo futuro

# Tabla de contenidos

# Capítulo 10. Conclusiones y trabajo futuro

## 10.1. Visión global del trabajo realizado

En el presente trabajo se ha realizado un sistema de gestión de una empresa mayorista del sector textil con la funcionalidad básica para poder ser usado en una empresa real.

Para el desarrollo del sistema se ha optado por soluciones innovadoras tanto en tecnología como metodología, escogiendo una de las llamadas ágiles como es *eXtreme Programming*, separando el proceso en sucesivas iteraciones que han añadido funcionalidad progresivamente, y que han permitido disponer de versiones de demostración cada pocas semanas.

La gestión de la configuración y documentación se ha realizado integralmente utilizando la novedosa herramienta Maven, que proporciona un alto nivel de estandarización de los proyectos, facilitando la integración de documentación, repositorio de código fuente, gestión de entregas, historial de cambios, gestión exhaustiva de versiones, porcentaje de código que está probado, y toda clase de herramientas para mejorar la gestión del proyecto.

La solución desarrollada ha proporcionado una arquitectura reutilizable, basada en patrones y tecnologías ampliamente utilizadas, que proporciona de forma sencilla características de seguridad y accesibilidad a toda la funcionalidad que construida sobre ella.

En cuanto a seguridad principalmente se destacan las características de gestión de autenticación y autorización trasparente, incluyendo, pero no limitado a, protección de urls, métodos u objetos, cifrado de contraseñas, redirección automática a canales seguros como SSL o características *single sign on* (autenticación para todas las aplicaciones de la empresa a través de un punto único). También relacionadas con la seguridad están las características de auditoría que permiten conocer en todo momento los cambios realizados por los usuarios.

En cuanto la accesibilidad, desde el núclo del sistema se ha dotado a los distintos módulos de un mecanismo automático para mostrar un interfaz basado en el navegador del usuario, centrado en proporcionar un interfaz adaptado a dispositivos móviles como PDAs. Ha de tenerse en cuenta que aunque el número de sitios web que ofrece contenido para estos dispositivos crece rápidamente, la práctica totalidad de ellos requieren que el usuario introduzca otra dirección web distinta a la habitual, por ejemplo Google ofrece su interfaz para PDAs a través de http://www.google.com/palm, mientras que la solución utilizada en este proyecto evita la necesidad de que el usuario necesite conocer otra dirección ya que la elección se hace automáticamente.

El aspecto del interfaz de usuario puede verse con las capturas de pantalla en la Figura 8.11, “Interfaz en Internet Explorer, página principal” y la Figura 8.12, “Interfaz en Internet Explorer,

resultado de una búsqueda" para el navegador web Internet Explorer y la Figura 8.13, "Interfaz en una Palm" para el navegador web de una Palm, así como ver una demostración interactiva del sistema en funcionamiento en la sección *demo* del sitio web en [Sourceforge].

El resultado del proyecto se ha puesto a disposición del público, según se ha ido realizando, como software *open source* certificado por la OSI (*Open Source Initiative*) bajo Apache License Version 2.0 a través del sitio web http://oness.sourceforge.net, donde se encuentra toda la documentación y los informes que acompañan al proyecto, todo ello en inglés para así poder llegar al mayor número de personas posible, utilizando toda una serie de mecanismos para fomentar su visibilidad como son listas de correo, repositorio de código fuente, weblog, seguimiento de incidencias,...

Como medida de éxito se exponen a continuación una serie de estadísticas.

**Figura 10.1. Estadísticas Sourceforge a fecha 18/9, gráfico**

En la Figura 10.1, "Estadísticas Sourceforge a fecha 18/9, gráfico" se pueden observar el número de páginas vistas, en rojo, y el número de descargas a través del interfaz de Sourceforge, en azul. El número concreto se puede ver en la Tabla 10.1, "Estadísticas Sourceforge a fecha 18/9". En este número de descargas no se incluyen ni las descargas de código fuente a través del repositorio CVS ni las que se realizan a través del repositorio de librerías que se ha creado para que sean accesibles desde cualquier proyecto gestionado con Maven.

**Tabla 10.1. Estadísticas Sourceforge a fecha 18/9**

| | Páginas vistas | Descargas |
|---|---|---|
| Abril | 350 | |
| Mayo | 2966 | |
| Junio | 3120 | |
| Julio | 9413 | 980 |
| Agosto | 14578 | 1402 |
| Septiembre | 11073 | 865 |
| **TOTAL** | **41500** | **3247** |

Como se puede ver el número tanto de descargas como de páginas vistas crece de manera importante. Teniendo en cuenta que a fecha 18/9 ya se supera la mitad de descargas que el mes

anterior se calcula que a finales de este mes se superarán las 50000 páginas y las 4000 descargas. Todos estos números hacen que el proyecto ocupe el puesto 200 en el ranking de más activos, que es un buen puesto teniendo en cuenta que Sourceforge es la mayor comunidad *open source* alojando 87673 proyectos y 919962 usuarios registrados según las últimas cifras conocidas, y que ha sido un proyecto unipersonal mientras que la mayoría de los otros proyectos cuentan con más de un desarrollador.

En cuanto al sitio web donde se ha instalado la demostración del proyecto los resultados se pueden ver en la Tabla 10.2, "Estadísticas de la aplicación de demostración del 14/7 al 18/9", contabilizados tan sólo a partir del 14 de julio, donde también se observa el crecimiento de visitas.

**Tabla 10.2. Estadísticas de la aplicación de demostración del 14/7 al 18/9**

| | Páginas vistas | Visitas |
|---|---|---|
| Julio | 4915 | 678 |
| Agosto | 10896 | 1391 |
| Septiembre | 6344 | 1014 |
| **TOTAL** | **22155** | **3083** |

Otra medida del éxito del proyecto es la proyección alcanzada por el autor, siendo invitado a incorporarse al proyecto Maven como desarrollador, entrando a formar parte de la mayor y más prestigiosa comunidad *open source* mundial, la *Apache Software Foundation*. También se puede tomar como referencia el número de visitas al weblog personal, donde se tratan temas relacionados tanto con el proyecto como con las tecnologías utilizadas, y que supera las 400 diarias. Es digno de comentar que en los resultados del sistema de búsqueda *Google* la web del proyecto ocupa el primer lugar cuando se busca por nombre y se encuentra entre las primeras posiciones cuando se buscan temas relacionados con las tecnologías empleadas.

Aparte de la empresa considerada desde el principio como objetivo de este proyecto, entre los usuarios se encuentra la compañía *Acegi Technology Pty Limited*, con base en Belmont, Australia, que utiliza el núcleo del proyecto como base para una aplicación web de comercio electrónico para la venta de piedras preciosas.

## 10.2. Aspectos favorables

Gestión metodológica del proyecto, con control e historial de cambios, gestión de versiones y documentación exhaustiva.

La arquitectura desarrollada es extensible y reutilizable, siendo realmente sencilla y rápida la implementación de nuevas funcionalidades.

Características de seguridad no intrusivas y totalmente configurables tal y como se ha mencionado en el apartado anterior.

Auditabilidad total de los cambios realizados por los usuarios, con la posibilidad de deshacerlos.

Accesibilidad transparente desde dispositivos móviles, también mencionada en el apartado anterior.

Utilización de patrones ampliamente extendidos, entre ellos el patrón *Model-View-Controller* que permite la separación de roles entre los desarrolladores.

El sistema está concienzudamente probado mediante tests automatizados, según los informes automáticamente generados por Maven el porcentaje de líneas de código probadas ronda el 85-90%.

Utilización en la medida de lo posible de soluciones recientes e innovadoras y con una amplia comunidad de usuarios, integrando tecnologías en lugar de realizar un desarrollo propio.

Gran éxito entre la comunidad *open source* tal y como se puede ver en las estadísticas anteriormente mostradas, con un creciente número de usuarios.

## 10.3. Aspectos desfavorables

El principal aspecto negativo ha sido el tiempo de aprendizaje requerido para utilizar las tecnologías empleadas. Como ya se ha indicado esto ha causado grandes retrasos en las primera fase del proyecto, si bien la mayoría de los problemas han sido causados por la utilización del proyecto Hibernate ya que esta solución de persistencia no se ajusta del todo bien a aplicaciones en capas que pretenden mantener una independencia entre las mismas, sino que está más pensada para proporcionar una persistencia transparente en aplicaciones monocapa o permitiendo que desde capas superiores se pueda acceder al sistema gestor de base de datos. A pesar de estos problemas la utilización de Hibernate supone grandes ventajas sobre otras tecnologías como EJB una vez superado este proceso de aprendizaje.

Otro aspecto que sería mejorable sería la utilización de otro framework para la realización del interfaz web en lugar de Tiles, como pueden ser Sitemesh o preferiblemente el reciente estándar JSF (*Java Server Faces*), que permitieran compartir recursos entre las aplicaciones web de cada módulo en momento de ejecución.

## 10.4. Trabajo futuro

Como trabajo inmediato se desplegará el sistema desarrollado en la empresa que ha facilitado los datos y la experiencia del sector, integrándolo con el sistema actual en funcionamiento.

En cuanto al proyecto ONess se abrirán las puertas a la incorporación de nuevos desarrolladores siguiendo una política basada en méritos similar a la utilizada en la *Apache Software Foundation*. El futuro desarrollo se orientará hacia la creación de otros componentes de negocio genéricos y reutilizables, con lo que se prevé un mayor interés por parte de desarrolladores de todo el mundo, interés ya demostrado por algunos de ellos.

Un proyecto ya propuesto que utilizará ONess como base será la aplicación web que gestionará el repositorio central de Maven, el mayor repositorio de librerías Java existente, y que pretende delegar la responsabilidad de añadir nuevas librerías a los desarrolladores de cada una de ellas, proceso que hasta el momento se ha realizado por parte de los desarrolladores de Maven, con lo que se requerirá una completa gestión de usuarios, permisos y auditoría de cambios, puntos fuertes de la arquitectura desarrollada por el proyecto ONess.

# Apéndice A. Obtención y compilación del código

## A.1. Descarga y compilación del código fuente disponible en CVS

El proyecto utiliza Maven [http://maven.apache.org] como herramienta de gestión. Tan sólo se necesita seguir los siguientes pasos:

1. Instalar Maven (al menos versión 1.0).

2. Ejecutar la siguiente instrucción en una única línea para obtener los últimos fuentes de CVS.

```
maven scm:checkout-project
  -Dmaven.scm.method=cvs
  -Dmaven.scm.cvs.module=.
  -Dmaven.scm.cvs.root=:pserver:anonymous@cvs.sourceforge.net:/cvsroot/oness
  -Dmaven.scm.checkout.dir=oness[1]
```

3. Una vez hecho se puede ejecutar maven en el directorio de cualquier módulo, o **`maven multiproject:artifact`** en el directorio `doc` para compilar todos los módulos y generar las librerías (ficheros jar) y las aplicaciones web (ficheros war).

## A.2. Compilación a partir del código fuente descargado

1. Obtener el módulo deseado a partir de la página web [Sourceforge] y descomprimirlo en un directorio, por ejemplo si se obtiene oness-party-model-src-0.2 descomprimirlo a un directorio undirectorio/party/model (el nombre no importa, sólo asegúrese de que hay dos niveles de directorios si el nombre del módulo es oness-x-y-version o tres si es oness-x-y-z-version)

2. Obtener oness-common-maven y descomprimirlo en undirectorio/common/maven (el

[1]Para obtener la versión concreta presentada como proyecto de fin de carrera se puede añadir **`-Dmaven.scm.cvs.sticky.tag=UDC`**

nombre es importante)

3. Ya se puede ejecutar maven en el directorio del módulo descargado, por ejemplo para compilar las clases java se puede utilizar **`maven java:compile`**, o **`maven jar`** para generar la librería jar. Consulte la documentación de Maven para saber todas las posibilidades que ofrece.

## A.3. Configuración de la base de datos para los tests

La configuración de la conexión a la base de datos por defecto está en el módulo oness-common-model, en el fichero `src/conf/dataSource.properties`, que debería servir para una instalación por defecto de MySQL.

Si se desea cambiar la base de datos, usuario o contraseña se puede añadir algunas de las siguientes propiedades en el fichero `$HOME/build.properties` que tendrán precedencia sobre los valores por defecto:

```
maven.junit.sysproperties=dataSource.password \
    dataSource.username \
    dataSource.url \
    dataSource.driverClassName \
    hibernate.dialect

# If you want to use postgres
dataSource.username=yourpostgresusername
dataSource.password=
dataSource.url=jdbc:postgresql:yourdatabasename
dataSource.driverClassName=org.postgresql.Driver
hibernate.dialect=net.sf.hibernate.dialect.PostgreSQLDialect
```

En caso de que no se desee ejecutar los test de unidad se puede ejecutar Maven con el parámetro **`-Dmaven.test.skip=true`**.

## A.4. Usando eclipse

Si desea utilizar eclipse necesitará instalar algunas librerías en el repositorio local de Maven, si no desea modificar los ficheros de proyecto de eclipse. Para realizarlo se debe llamar a **`maven jar:install-snapshot`** en los módulos que necesite o **`maven multiproject:install-snapshot`** en el módulo `doc` para instalar las librerías de todos los módulos.

# Apéndice B. Instalación

## B.1. Instalación

La instalación del sistema completo implica la instalación de una base de datos y tres aplicaciones web, `oness-party-webapp.war`, `oness-inventory-webapp.war` y `oness-order-webapp.war`, en un contenedor web o servidor de aplicaciones. Las instrucciones concretas varían para cada servidor, por lo que se pondrán como ejemplo las correspondientes a la implementación de referencia Tomcat.

## B.2. Instalación de la base de datos

El funcionamiento de la siguientes bases de datos ha sido probado con el proyecto.

- MySQL 4.0.13-Max bajo Windows
- MySQL 4.0.15-Max bajo Linux
- PostgreSQL 7.4.5 bajo Cygwin en Windows
- MS SQL 2000
- HSQLDB 1.7.2, esta versión **NO** es compatible con el software dado que no soporta niveles de aislamiento entre transacciones.

Culquiera de ellas excepto HSQLDB podría ser usada así como cualquier otra soportada por Hibernate, en cuyo sitio web http://www.hibernate.org/80.html se puede encontrar la lista de compatibilidades. Entre los sistemas soportados se encuentran, además de los anteriores, otros como DB2, Oracle, Informix o Sybase.

## B.3. Instalación en Tomcat

La instalación en Tomcat 4.x o 5.x conlleva copiar los ficheros war en el directorio `TOMCAT_HOME/webapps` y configurar la base de datos que se utilizará. Por defecto esta configuración se obtendrá via JNDI, con lo que no será necesario modificar los ficheros war.

En el entorno JNDI deben estar disponibles una variable de entorno con nombre `net.sf.oness.common.model.hibernateDialect` y cuyo valor debe ser el dialecto de Hibernate para el gestor de base de datos utilizado, y una fuente de datos bajo el nombre

`jdbc/oness` de donde obtener las conexiones con la base de datos. Para configurar el entorno JNDI en Tomcat se debe crear un fichero xml para cada war y con el mismo nombre en el directorio `TOMCAT_HOME/webapps`, si se trata de Tomcat 4.x, o `TOMCAT_HOME/conf/Catalina/localhost`, si la versión es 5.x. El contenido de cada uno de estos ficheros xml se puede ver en el Ejemplo B.1, “Configuración de la aplicación web en Tomcat utilizando JNDI”. También podría utilizarse una configuración global para todas las aplicaciones, lo que implicaría modificar el fichero de configuración del servidor, por lo que no será tratada aquí, pudiendo consultarse la documentación de Tomcat.

**Ejemplo B.1. Configuración de la aplicación web en Tomcat utilizando JNDI**

```
<!--

    To setup this context copy this file to the specified
    directory and change it to match your needs.
    You may rename it if you want.

    o Tomcat 4: TOMCAT_HOME/webapps
    o Tomcat 5: TOMCAT_HOME/conf/Catalina/localhost

-->

<Context path="/oness-party-webapp" ❶
    docBase="${catalina.home}/webapps/oness-party-webapp.war" ❷
    debug="99"
    reloadable="true">

    <!--
        To use a global jndi datasource uncomment the <ResourceLink> tag
        and move the other entries to your server.xml
        under <GlobalNamingResources>
    -->
    <!--
    <ResourceLink name="jdbc/oness"
        global="jdbc/oness"
        type="javax.sql.DataSource"/>
    -->

    <Environment description="Hibernate dialect"
        name="net.sf.oness.common.model.hibernateDialect"
        value="net.sf.hibernate.dialect.MySQLDialect" ❸
        type="java.lang.String"/>

    <Resource name="jdbc/oness"
        auth="Container"
        type="javax.sql.DataSource"/>

    <ResourceParams name="jdbc/oness">
        <parameter>
            <name>factory</name>
            <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
        </parameter>
        <!-- Maximum number of dB connections in pool. Make sure you
             configure your mysqld max_connections large enough to handle
             all of your db connections. Set to 0 for no limit.
             -->
```

```
        <parameter>
            <name>maxActive</name>
            <value>100</value>
        </parameter>
        <!-- Maximum number of idle dB connections to retain in pool.
             Set to 0 for no limit.
             -->
        <parameter>
            <name>maxIdle</name>
            <value>30</value>
        </parameter>
        <!-- Maximum time to wait for a dB connection to become available
             in ms, in this example 10 seconds. An Exception is thrown if
             this timeout is exceeded.  Set to -1 to wait indefinitely.
             -->
        <parameter>
            <name>maxWait</name>
            <value>10000</value>
        </parameter>
        <!-- MySQL dB username and password for dB connections  -->
        <parameter>
            <name>username</name> ❹
            <value></value>
        </parameter>
        <parameter>
            <name>password</name> ❺
            <value></value>
        </parameter>
        <!-- Class name for JDBC driver -->
        <parameter>
            <name>driverClassName</name>
            <value>com.mysql.jdbc.Driver</value> ❻
        </parameter>
        <!-- Autocommit setting.  This setting is required to make
             Hibernate work.  Or you can remove calls to commit(). -->
        <parameter>
            <name>defaultAutoCommit</name>
            <value>false</value>
        </parameter>
        <!-- The JDBC connection url for connecting to your MySQL dB.
             The autoReconnect=true argument to the url makes sure that the
             mm.mysql JDBC Driver will automatically reconnect if mysqld closed the
             connection.  mysqld by default closes idle connections after 8 hours.
             -->
        <parameter>
            <name>url</name>
            <value>jdbc:mysql://localhost/test</value> ❼
        </parameter>
        <!-- Recover abandoned connections -->
        <parameter>
            <name>removeAbandoned</name>
            <value>true</value>
        </parameter>
        <!-- Set the number of seconds a dB connection has been idle
             before it is considered abandoned.
             -->
        <parameter>
            <name>removeAbandonedTimeout</name>
            <value>60</value>
        </parameter>
        <!-- Log a stack trace of the code which abandoned the dB
```

```
             connection resources.
             -->
        <parameter>
            <name>logAbandoned</name>
            <value>true</value>
        </parameter>
    </ResourceParams>

</Context>
```

❶ Url en la que estará disponible la aplicación dentro del servidor
❷ Ubicación del fichero war
❸ Dialecto de Hibernate para el gestor de base de datos utilizado
❹ Usuario de la base de datos
❺ Contraseña de la base de datos
❻ Clase del driver JDBC para el gestor de base de datos
❼ Url de la base de datos

La alternativa a utilizar JNDI pasaría por modificar el descriptor de aplicación web `web.xml` dentro del fichero war, cambiando el valor `applicationContext-oness-common-model-dataSource-jndi.xml` por `applicationContext-oness-common-model-dataSource-dbcp.xml`, y añadiendo al classpath un fichero llamado `dataSource.properties` con mayor precedencia que el fichero `oness-common-model-x.x.jar`, por ejemplo ubicándolo en el directorio `classes` de la aplicación web. El contenido de este fichero para configurar una base de datos MySQL se puede ver en el Ejemplo B.2, “Propiedades de la fuente de datos DBCP para MySQL”.

**Ejemplo B.2. Propiedades de la fuente de datos DBCP para MySQL**

```
dataSource.username=❶
dataSource.password=❷
dataSource.url=jdbc:mysql:///test❸
dataSource.driverClassName=com.mysql.jdbc.Driver❹
hibernate.dialect=net.sf.hibernate.dialect.MySQLDialect❺
```

❶ Nombre de usuario en la base de datos
❷ Contraseña en la base de datos
❸ URL de la base de datos, depende de cada gestor
❹ Nombre de la clase controladora JBDC, depende de cada gestor
❺ Dialecto del gestor de la base de datos, proporcionado por Hibernate

# Glosario

API

Application Programming Interface. Especificación de una librería o utilidad que documenta su interfaz y permite su uso sin conocimiento de su interior.

ASF

Apache Software Foundation, http://www.apache.org. Fundación que proporciona soporte a proyectos *open source*, formando una comunidad de desarrolladores y usuarios con una gran reputación por la calidad y el éxito de muchos de sus proyectos. Dentro de la comunidad Java participa activamente, formando parte de los comités estandarizadores.

CSS

Cascading Style Sheets. Tecnología que permite crear páginas web con un diseño más exacto, añadiendo mayores posibilidades a HTML y permitiendo una mayor separación entre la información y la presentación.

Framework

Conjunto de APIs y herramientas destinadas a la construcción de un determinado tipo de aplicaciones de manera generalista.

HTML

Hypertext Markup Language. Lenguage de tags estandarizado para la creación de documentos para la web.

HTTP

Hypertext Transfer Protocol. Protocolo de nivel de aplicación usado extensivamente en Internet para el acceso a documentos.

ISO

Federación de alcance mundial integrada por cuerpos de estandarización nacionales de gran número de países, que promueve el desarrollo de la estandarización.

J2EE

Java 2, Enterprise Edition. Versión avanzada de la plataforma Java de Sun Microsystems, destinada al desarrollo de aplicaciones empresariales.

J2SE

| | |
|---|---|
| | Java 2, Standard Edition. Versión básica del conjunto de herramientas y APIs de Sun Microsystems destinadas a la creación de aplicaciones en plataforma Java. |
| JAAS | Java Authentication and Authorization Specification. Conjunto de APIs que permite que servicios puedan autenticar e imponer controles de acceso a los usuarios. |
| Java | Plataforma para el desarrollo de software creada por Sun Microsystems, ampliamente extendida hoy en día, que otorga independencia de plataforma al software creado en ella y lo provee de una gran cantidad de APIs estandarizados. |
| JavaDoc | Documentación HTML embebida en el código Java de una aplicación, que mediante la herramienta del mismo nombre puede ser convertida en un árbol de páginas HTML que muestre la documentación del API de la aplicación de manera detallada. |
| JDBC | Java DataBase Connectivity. API de la plataforma Java para el acceso a sistemas gestores de base de datos. |
| JNDI | Java Naming and Directory Interface. Servicio estándar de nombrado y directorio en Java. |
| JSP | JavaServer Pages. API de la plataforma J2EE para la creación de páginas web dinámicas mediante el uso de librerías de tags. |
| JSTL | JavaServer Pages Standard Tag Library. Especicación J2EE de una serie de librerías de tags de propósito general para páginas JSP que aumenten su funcionalidad. |
| Model2 | Modelo arquitectónico para aplicaciones web acuñado por Sun Microsystems consistente en la creación de una aplicación según el patrón MVC y estableciendo un controlador único o front controller en su capa controlador. |

MVC

Model-View-Controller. Patrón arquitectónico desarrollado para interfaces grácas que resalta la importancia de una separación clara entre la presentación de datos y la lógica de negocio de una aplicación.

open source

Calificación de software que cumple una serie de requisitos, principalmente aquel que permite una libre redistribución, distribuye el código fuente, y permite modificaciones y trabajos derivados.

OSI

Open Source Initiative, http://www.opensource.org. Organización sin ánimo de lucro dedicada a la gestión y promoción del software *open source*, específicamente a través del programa de certificación de software *open source*.
Ver también open source.

SSL

Secure Socket Layer. Protocolo que proporciona servicios de seguridad cifrando los datos intercambiados entre el servidor y el cliente.

UML

Unified Modeling Language. Lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software.

weblog

Página web estilo diario que facilita un interfaz por el que los usuarios pueden recibir las noticias expuestas en él de manera automática.

XML

eXtensible Markup Language. Formato estándar para el intercambio de datos basado en archivos de texto plano con una estructura de tags.

XP

eXtreme Programming. Metodología de desarrollo de software basada en valores como simplicidad, comunicación, retroalimentación y coraje.

# Bibliografía y Referencias

[Sourceforge] *Sitio web en Sourceforge*. http://oness.sourceforge.net .

# Tecnologías

[Laddad03] *AspectJ in Action*. Ramnivas Laddad. 2003. Manning.

[JohnsonHoeller04] *Expert One-on-One J2EE Development without EJB*. Rod Johnson y Juergen Hoeller. 2004. Wrox.

[Johnson02] *Expert One-on-One J2EE Design and Development*. Rod Johnson. 2002. Wrox.

[TateGehtland04] *Better, Faster, Lighter Java*. Justin Gehtland y Bruce A. Tate. 2004. O'Reilly.

[Chan03] *Project management: Maven makes it easy*. http://www-106.ibm.com/developerworks/java/library/j-maven/ . Charles Chan. 2003.

[O'Regan04] *Introduction to Aspect-Oriented Programming*. http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html . Graham O'Regan. 2004.

[CAS] *Yale University's Central Authentication Service*. http://www.yale.edu/tp/auth/ . Yale University.

[HustedDumoulinFranciscusWinterfeldt03] *Struts in Action*. Manning Publications. 2003. Ted Husted, Cedric Dumoulin, George Franciscus, y David Winterfeldt.

[Bayern03] *JSTL in Action*. Manning Publications. 2003. Shawn Bayern.

[Cavaness02] *Programming Jakarta Struts*. O'Reilly. 2002. Chuck Cavaness.

[Turner] *Struts indexed properties and validation*. James Turner. http://www.strutskickstart.com .

# Arquitectura

[AlurCrupiMalks03] *Core J2EE Patterns*. Second Edition. Prentice Hall. 2003. Deepak Alur, John Crupi, y Dan Malks.

[SinghStearnsJohnson02] *Designing Enterprise Applications with the J2EE Platform*. Second Edition. Addison Wesley. 2002. Inderjeet Singh, Beth Stearns, y Mark Johnson.

# Metodología

[Fowler03] *The New Methodology*. http://www.martinfowler.com/articles/newMethodology.html . Martin Fowler. 2003.

[AgileManifesto] *Manifesto for Agile Software Development* http://agilemanifesto.org Kent Beck, James Grenning, Robert C. Martin, et. al. 2001.

[Wells03] *Extreme Programming: A gentle introduction.*. http://www.extremeprogramming.org . Don Wells. 2003.

[Jeffries01] *What is Extreme Programming?*. http://www.xprogramming.com/xpmag/whatisxp.htm . Ron Jeffries. 2001.

[Ferrer03] *Metodologías Ágiles*. http://libresoft.dat.escet.urjc.es/html/downloads/ferrer-20030312.pdf . Jorge Ferrer Zarzuela. 2003.

[CanosLetelierPenades03] *Métodologías Ágiles en el Desarrollo de Software*. http://www.willydev.net/descargas/prev/TodoAgil.Pdf . José H. Canós, Patricio Letelier, y Mª Carmen Penadés. 2003.

[LetelierPenades03] *Métodologías Ágiles para el desarrollo de software: eXtreme Programming (XP)*. http://www.willydev.net/descargas/masyxp.pdf . Patricio Letelier y Mª Carmen Penadés. 2004.

[JavaCodeConventions] *Java Code Conventions*. http://java.sun.com/docs/codeconv/index.html . Sun Microsystems.

# Patrones de diseño temporal

[Fowler] *Analysis Patterns 2*. http://www.martinfowler.com/ap2/index.html . Martin Fowler. Trabajo en progreso.

[CarlsonEsteppFowler99] *Pattern Languages of Program Design 4*. Addison-Wesley. Foote Harrison. 1999. “Temporal Patterns”. Andy Carlson, Sharon Estepp, y Martin Fowler. 241-262.

[NogueiraEdelweiss00] *XV Simpósio Brasileiro de Banco de Dados – SBBD'2000*. 2000. “Implementing a Temporal Database on Top of a Conventional Database: Mapping of the Data Model and Data Definition Management”. Patrícia Nogueira Hübler y Nina Edelweiss.

# Misceláneo

[OMGParty01] *Party Management Facility specification*. http://www.omg.org/technology/documents/formal/party_mgmt.htm . OMG. 2001.

[Wheeler04] *Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!*. http://www.dwheeler.com/oss_fs_why.html . David A. Wheeler. 2004.