Estudio, Comparativa y Aplicación Práctica de Metodologías de Desarrollo de Aplicaciones Web en Java

Francisco José García Ruiz
Ingeniería en Informática
Javier Ferro García
14 de Enero de 2008

RESUMEN

El presente proyecto se enmarca dentro del área J2EE y pretende servir de ayuda para de definir la arquitectura, la estructura, la metodología y las herramientas más adecuadas para el desarrollo de una aplicación Web Java. En este sentido presenta un estudio sobre las arquitecturas, patrones de diseño y frameworks java más relevantes. Con este estudio se pretende plasmar un conjunto de conceptos que son básicos en el desarrollo de aplicaciones Web desacopladas, mantenibles, escalables, fácilmente desplegables etc.

El estudio se centra sobre algunos de los frameworks que podemos encontrar en el mercado. Además, agrupa los frameworks en función de la parte de la aplicación sobre la que centra su soporte, presentación, lógica de negocio, persistencia de datos, seguridad…, ofreciendo una visión de sus aspectos más relevantes. A continuación se realiza una comparación entre algunos de los frameworks estudiados para facilitar su elección según las necesidades de la aplicación a construir.

El último punto del proyecto consiste en realizar una aplicación práctica sobre del estudio realizado. En este sentido se presenta un caso del mundo real para el que es necesario implementar una solución. Ésta pasa por una elección justificada de la arquitectura, frameworks etc. que van a usarse de soporte. Este punto pretende dar información detallada sobre la configuración e integración de diferentes frameworks para conseguir una aplicación robusta que cumple con las necesidades presentadas.

1 INTRODUCCIÓN

1.1 Justificación y Contexto

La constante, casi frenética, evolución que está sufriendo el desarrollo de Internet y el aumento del número de usuarios que lo utiliza, está causando un gran auge en el desarrollo de aplicaciones Web. Este tipo de software permite que un determinado usuario pueda acceder a información almacenada en servidores Web a través de Internet o de una intranet. El factor clave radica en la posibilidad de poder acceder a una determinada aplicación, de forma transparente[1] al usuario y desde cualquier lugar. Este valor añadido hace que muchas empresas opten, por ejemplo, realizar intranets[2] corporativas, permitiendo el despliegue de aplicaciones Web para compartir y gestionar información.

Desde el punto de vista de los profesionales que se dedican al mundo de la informática y en particular, al desarrollo de aplicaciones Web, la creciente evolución del sector conlleva la mejora e innovación de los lenguajes y herramientas disponibles para desarrollar este tipo de software. Además, muchos de los problemas que se presentan en el desarrollo de aplicaciones Web no existen en las aplicaciones de escritorio y se debe, fundamentalmente, a que el protocolo HTTP, en el cual se basa la comunicación cliente-servidor, es un protocolo sin estado. Con lo que cada vez más, aparecen nuevas herramientas que hacen que la construcción y diseño de un entorno Web sea más fácil y rápido de hacer. En un mundo competitivo como en el que vivimos donde todo tiene que estar al momento ("just in time"), son de agradecer las herramientas de desarrollo que siguen los conocidos patrones de diseño y que permiten implementar aplicaciones de una manera rápida y estructurada. Pero no solo eso, sino que además de todo se desea que las estructuras de las aplicaciones sean extensibles, es decir, no nos sirve de nada que empleemos cientos de horas en fijar los cimientos de un proyecto si luego éstos no nos van a servir para uno futuro. Este proyecto pretende repasar un conjunto de técnicas y herramientas que nos pueden ser de utilidad en el desarrollo de una aplicación Web. Además se presentará un caso práctico para plasmar algunos de los conceptos repasados.

1.2 Punto de partida

El punto de partida del presente proyecto radica en la elección del lenguaje de programación y el tipo de arquitectura sobre el que pretendemos realizar el estudio y la implementación de una aplicación de ejemplo. Esta decisión condicionará la elección del entorno de desarrollo y las tecnologías usadas para la implementación de la aplicación. En nuestro caso, el lenguaje elegido para realizar el desarrollo, y por tanto todo el estudio previo, es Java. El motivo de esta elección se fundamenta en que es un lenguaje altamente extendido, independiente de la plataforma hardware/software, orientado a objetos, con gran cantidad de herramientas que dan soporte a su programación y además es relativamente sencillo de utilizar y mantener. A partir de aquí nos centraremos en el estudio y selección de un conjunto de frameworks J2EE de código libre. La arquitectura final de aplicación Web depende de la separación en capas y partes que se decida, así como de los patrones de diseño utilizados por las diferentes (una o varias) herramientas que se unen para el desarrollo del producto final.

1.3 Descripción del trabajo a realizar

Uno de los aspectos más interesantes de Java es el ritmo al que evolucionan tanto sus tecnologías como las arquitecturas diseñadas para soportarlas. Cada poco tiempo aparecen nuevas librerías, herramientas, frameworks, patrones, etc., y se hace muy difícil elegir entre ellos y, sobre todo, combinarlos de una manera eficiente que aproveche toda su flexibilidad y potencia. En este sentido, La primera tarea que debe realizarse es la búsqueda de información referente a los diferentes técnicas de diseño y framewoks existentes en el mercado para dar soporte a la implementación de aplicaciones Web en Java siguiendo diferentes patrones. En este sentido, el estudio se centrará en herramientas de código abierto. La elección de este tipo de herramientas se fundamenta en la flexibilidad, flexibilidad, seguridad, rapidez de desarrollo, libre

[1] No es necesario realizar ningún tipo de instalación en el ordenador que accede como cliente. Es suficiente con que disponga de un navegador Internet.

[2] Redes privadas de ordenadores que permiten el desarrollo de aplicaciones Web para compartir aplicaciones y datos.

distribución, que este tipo de software aporta al desarrollo de una aplicación Web. Se contemplarán frameworks que den soporte a la implementación de las diferentes capas que puede presentar una aplicación Web (capa de presentación, lógica de negocio, persistencia de información, seguridad...).

A continuación se realizará un análisis detallado de las herramientas seleccionadas desde el punto de vista funcional (más técnico) y no funcional (eficiencia, seguridad, facilidad de mantenimiento, facilidad de programación). Además se evaluará su capacidad de implementación de las diferentes capas del modelo los diferentes modelos arquitectónicos. Finalmente se realizará una comparación entre las diferentes tecnologías con la finalidad de que pueda servir de ayuda en la elección de las herramientas de desarrollo concretas.

Seguidamente se seleccionará un modelo arquitectónico a partir del estudio previo realizado. Se justificará cada una de las herramientas seleccionadas con respecto a otras existentes para implementar las diferentes capas. A partir de esta elección se implementará una pequeña aplicación de ejemplo que sirva de muestra para ver como se integran estas tecnologías en una aplicación Web.

1.4 Objetivos

Este proyecto pretende, dar una visión detallada de los principales frameworks existentes para el desarrollo de aplicaciones Web de código abierto (de distribución y desarrollo libre) basados en Java. A partir del estudio realizado, se pretende aportar información útil para ayudar a seleccionar el conjunto de herramientas adecuadas que van a ser utilizadas como soporte en el desarrollo de una aplicación Web. A partir de la elección justificada de una arquitectura de ejemplo, se pretende ejemplarizar su uso e integración de sus diferentes herramientas en una pequeña aplicación de ejemplo.

1.5 Planificación del trabajo a realizar

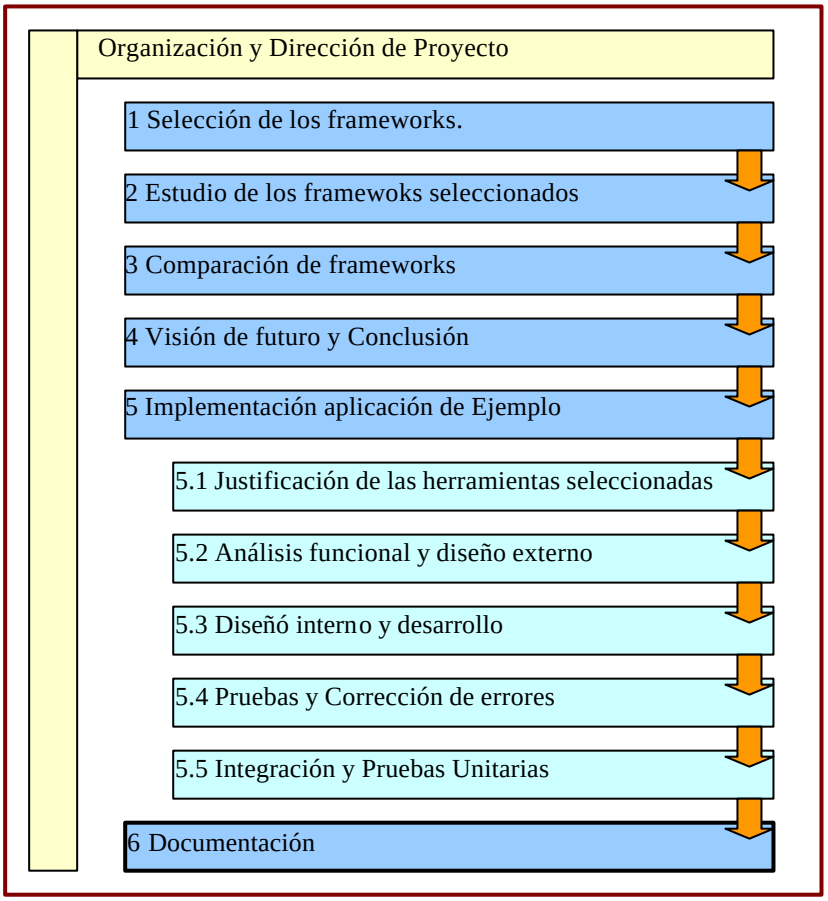
1.5.1 Definición de Fases

A efectos de control y seguimiento, se considera el proyecto, desde el punto de vista de su estudio e implementación, dividido en 6 grandes áreas:

- Fase 1: Selección de los frameworks, en ella se pretende justificar su uso para el desarrollo de aplicaciones Web así como recopilar información relacionada con algunos de ellos. Las herramientas seleccionadas serán básicamente de código abierto, y capaces de dar soporte a aplicaciones Java. La selección se realizará de forma justificada.
- Fase 2: Estudio de los frameworks seleccionados, en ella se pretende mostrar los aspectos más importantes tanto a nivel funcional como no funcional. Principales ventajas e inconvenientes en el momento de elegirlos para el desarrollo de una aplicación.
- Fase 3: Comparación de frameworks, en ella se pretende realizar un estudio comparativo entre las diferentes tecnologías seleccionadas. La comparación se realizará a partir de ventajas e inconvenientes y de su capacidad de dar soporte a cada una de las capas de una aplicación Web.
- Fase 4: Visión de futuro y Conclusión, en ella se darán una aproximación a la evolución futura de dichas herramientas. Se intentará presentar la tendencia de futuro de este tipo de herramientas. Finalmente se presentará una conclusión del estudio realizado.
- Fase 5: Implementación aplicación de Ejemplo, en ella se pretende ofrecer un ejemplo práctico de utilización de un conjunto de herramientas seleccionadas. Dentro de esta fase y enmarcada en el desarrollo de una aplicación se contemplan las siguientes subtareas:
    - Justificación de las herramientas seleccionadas, breve justificación de la arquitectura seleccionada para realizar la aplicación. Se justificará la elección de las diferentes herramientas.
    - Análisis funcional y diseño externo, el objetivo del presente proyecto no es entrar en el detalle del desarrollo de una aplicación Web, por tanto esta fase presentará una visión a alto nivel de la funcionalidad del sistema. Como resultado de esta fase se obtendrá una breve descripción de especificaciones funcionales y el diseño de la base de datos.
    - Diseño interno y desarrollo, se prepara el entorno de desarrollo y se realiza la documentación del sistema a partir de la documentación de la fase anterior.

- Pruebas y corrección de errores, se realizarán pruebas para corregir posibles incidencias.
- Integración y Pruebas Unitarias, se acoplan todos los componentes de la aplicación y se realizan pruebas globales del sistema.

- Fase 6: Documentación, en ella se pretende completar toda la información relativa al proyecto.



De forma simultánea a todas las fases del proyecto se realizarán tareas de organización y dirección del mismo. Dichas tareas abarcan la totalidad del proyecto y no finalizan hasta que éste lo haga. Se debe mantener la coordinación entre todas las partes, realizando el seguimiento del plan y calendario definidos.

1.5.2 Cronograma de fases

A continuación se presenta un gráfico que muestra una estimación en semanas del tiempo necesario para la realización del proyecto así como la distribución de las diferentes tareas.

CRONOGRAMA DEL PROYECTO																			
	SEMANAS DESDE EL INICIO DEL PROYECTO																		
	SEPTIEMBRE				OCTUBRE				NOVIEMBRE					DICIEMBRE				ENERO	
	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19
FASE 1																			
FASE 2																			
FASE 3																			
FASE 4																			
FASE 5																			
FASE 5.1																			
FASE 5.2																			
FASE 5.3																			
FASE 5.4																			
FASE 5.5																			
FASE 6																			

1.5.3 Capítulos de la Memoria

Inicialmente se realiza un repaso por el concepto de patrón. Se trata de un punto de partida básico prévio al diseño de cualquier aplicación. Es fundamental tener claro las técnicas disponibles que nos faliliten el diseño e implementación de una determinada aplicación. En este sentido realizaremos un breve repaso del concepto de patrón desde el punto de vista arquitectónico y de diseño, para finalmente resaltar los patrones más representativos del mundo Java.

Centrando ya el estudio completamente en el mundo Java, el siguiente capítulo realiza un breve repaso del concepto de especificación diseñado por Sun que permiten la creación de aplicaciones empresariales unificadas y compatibles para un determinado entorno de ejecución.

A continuación, el estudio se posiciona en las herramientas de soporte al desarrollo de aplicaciones Web, o sea los frameworks. El análisis se organiza según la capa de la aplicación sobre la que se centra cada framework, o sea, vista, lógica, persistencia, seguridad, etc. Se realiza una selección de los más representativos, describiendo brevemente algunas de sus características más representativas.

En el siguiente punto se realiza una selección de algunos de los frameworks anteriores para cada una de las capas, basado en criterios de uso por la comunidad, documentación existente, calidad del proyecto, etc. El estudio entra en un mayor detalle de cada uno de ellos para realizar un análisis comparativo con el fin de facilitar la elección de uno u otro en función de las necesidades del sistema.

El estudio finaliza con una aplicación práctica de los conceptos repasados hasta este punto. El objetivo es presentar una aplicación del mundo real donde el lector pueda ver como aplican las técnicas y herramientas estudiadas.

2 PATRONES

2.1 ¿Que es un patrón?

Uno de los aspectos más importantes en el desarrollo de sistemas es la experiencia. En este sentido, uno de los mayores avances de que se dispone, es la información recompilada a lo largo del tiempo sobre problemas que se han ido presentando a diferentes personas de la comunidad. Más importante aún que la recopilación de problemas, es la manera correcta de solucionarlos. Pues bien, este planteamiento de formalizar soluciones a distintas situaciones, de modo que puedan ser entendidas por otros profesionales, es lo damos en llamar patrones. Por tanto, un patrón no es más que la descripción detallada de una solución adecuada a un problema concreto. Normalmente, un patrón está compuesto por el enunciado del problema y una o varias propuestas de solución. Para formalizar estas soluciones, se necesita una notación expresiva y rica en semántica que permita plasmar eficientemente los atributos de las mismas. Además esta notación ha de ser fácil de entender e interpretar. La notación más extendida es UML (Unified Modelling Language) que permite capturar con precisión las soluciones a problemas de todo tipo.

Junto a los patrones, aparecen también los antipatrones, que explican lo que nunca se debe hacer ante un problema dado. Son de gran ayuda, pues permiten esquivar errores que de otra manera serían muy fáciles de cometer.

2.2 Tipos de Patrón

Existen varios tipos de patrones, dependiendo del nivel de abstracción, del contexto particular en el cual aplican o de la etapa en el proceso de desarrollo. Algunos de estos tipos, de los cuales existen publicaciones el al respecto son:

- Patrones **organizativos**: Describen la estructura y prácticas de las organizaciones humanas, especialmente las productoras de software.
- Patrones de **análisis**: Describen un conjunto de prácticas destinadas a elaborar el modelo y solución de un problema. Se usan para construir modelos conceptuales de software.
- Patrones de **arquitectura**: Describen la estructura general de un sistema. Identifica sus módulos, responsabilidades, y la colaboración entre dichos módulos.
- Patrones de **diseño**: Definen aspectos como interfaces, componentes, distribución, extensión.
- Patrones de **programación** (Idioms): Pretenden describir como implementar ciertas tareas usando un lenguaje de programación concreto.
- Patrones de **seguridad**: describen mecanismos de seguridad.
- Patrones de **Integración de Aplicaciones**: En el mundo actual de la informática, pocos sistemas viven en solitario. Cada vez más es la regla, encontrarse con aplicaciones disímiles, de distintos fabricantes, en distintas plataformas, y que no se conectan entre sí. Aquí es donde este tipo de patrones pretenden facilitar las tareas de integración entre aplicaciones. La obra más popular al respecto es Enterprise Integration Patterns, de Gregor Hophe y Bobby Woolf.

En los siguientes puntos realizaremos una pequeña explicación sobre aspectos generales de los patrones más importantes para la realización de aplicaciones distribuidas. Finalmente realizaremos una breve descripción de algunos de los patrones de diseño más relevantes asociados a J2EE.

3 PATRONES ARQUITÉCTONICOS

Las aplicaciones Web se han convertido en pocos años en complejos sistemas con interfaces de usuario cada vez más parecidas a las aplicaciones de escritorio, dando servicio a procesos de negocio de considerable envergadura y estableciéndose sobre ellas requisitos estrictos de accesibilidad y respuesta. Esto exige reflexiones sobre la mejor arquitectura y las técnicas de diseño más adecuadas.

A continuación se realizará una breve descripción de las arquitecturas más comunes que podemos encontrarnos en este tipo de aplicaciones.

3.1 Arquitectura en 3 capas (layers)

En este tipo de arquitectura el reparto de responsabilidades es, en primera instancia, lógico. Y aunque posiblemente también sea físico, eso no es obligatorio. Las tres capas son:

- **Presentación**, recibe eventos del usuario a través de la interfaz presentada, y también formatea los resultados a desplegar.
- **Negocio**, el dominio del problema de negocio por el cual se tiene que desarrollar la aplicación
- **Acceso a Datos**, lógica que lleva a trae información entre la capa de negocio y los repositorios o sistemas externos donde los datos se almacenan (conectores, pools de conexiones, lógica de paginado, cachés...).

Aplicando este estilo arquitectónico se consigue mejorar la mantenibilidad y reusabilidad *(soporte a cambios tecnológicos en una capa).*

3.2 Arquitectura en 3 partes (tiers)

Aquí sí existe una separación física en procesos, es decir, la ejecución está distribuida. Las tres partes son:

- **Cliente o Front-End**, que engloba la infraestructura de hardware y software donde se ejecuta la interfaz de usuario. También se suele referir a ellos como Canales.
- **Middleware**, capaz de recibir a través de la red, y mediante uno o varios protocolos de transporte (HTTP, TCP, etc.), uno o varios protocolos de mensajes (XML, SOAP, propietarios, etc.) requerimientos desde los distintos canales, habilitándose así el concepto de Arquitectura Multicanal
- **Back-End**, normalmente una base de datos, aunque definitivamente puede ser otro proceso, incluso mucho más complejo que nuestra aplicación entera. Por ejemplo, un sistema mainframe con módulos de negocio. En ese caso nuestra aplicación es apenas si un habilitador de canales. Canales móviles con hardware diferente (PDA WiFi, etc.) y procesos diferentes (aplicaciones de escritorio, aplicaciones embebidas, aplicaciones web, etc.)

Normalmente las motivaciones que llevan a aplicar este tipo de arquitectura tienen que ver con Escalabilidad, Seguridad y Tolerancia a fallos. Definitivamente también este tipo de arquitectura habilita rehúso, ahora a nivel de procesos.

3.3 Arquitectura en n Capas y m Partes

Una aplicación puede tener, a la vez, Arquitectura en n capas y en m partes, sean n y m igual a 3 o no. No se trata de uno o el otro. Es eventualmente uno indistintamente de la presencia o no del otro. Algunas de las capas que podemos encontrar son:

- La capa del **cliente** donde se consumen y presentan los modelos de datos. Para una aplicación Web, la capa cliente normalmente es un navegador Web. Los clientes pequeños basados en navegador no contienen lógica de presentación; se trata en la capa de presentación.

- La capa de **presentación** está contenida, parte en el navegador cliente (lógica en JavaScript...), parte en el middleware servidor (páginas Velocity, ASP.NET, JSP, JSF, etc.). Sabe cómo procesar una petic ión de cliente, cómo interactuar con la capa de lógica de negocio, y cómo seleccionar la siguiente vista a mostrar.
- La capa de **negocio** pueden ser clases simples (los POJOs de Java o sus equivalentes POCOs de .NET - la C es por CLR, la máquina virtual de .NET), o clases complejas como EJBs, Serviced Components. Esta capa, puede ejecutarse en el mismo middleware o, por cuestiones de rendimiento, parte de la lógica de negocio se puede replicar en otras partes (tiers). Comúnmente, contiene los objetos y servicios de negocio de la aplicación. Recibe peticiones de la capa de presentación, procesa la lógica de negocio basada en las peticiones, y media en los accesos a los recursos de la capa EIS (persistencia). Sus componentes se benefician de la mayoría de lo servicios a nivel de sistema como el control de seguridad, de transacciones y de recursos.
- La capa de **integración** es el puente entre la capa de lógica de negocio y la capa EIS. Encapsula la lógica para interactuar con la capa EIS. Algunas veces a la combinación de las capas de integración y de lógica de negocio se le conoce como capa central.
- Los datos de la **aplicación** persisten en la capa EIS. Contiene bases de datos relacionales, bases de datos orientadas a objetos, y sistemas antiguos.

Por tanto, una arquitectura multicapa particiona todo el sistema en distintas unidades funcionales, Esto asegura una división clara de responsabilidades y hace que el sistema sea más mantenible y extensible. Los sistemas con tres o más capas se han probado como más escalables y flexibles que un sistema cliente / servidor, en el que no existe la capa central de lógica de negocios.

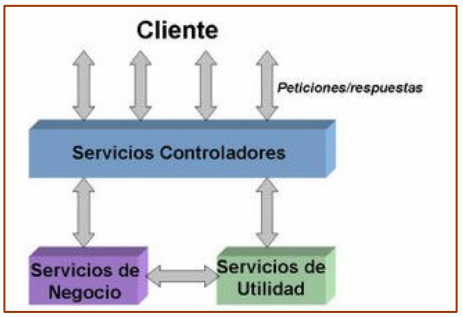
Aquí es donde entran a jugar las diferentes herramientas y frameworks que podemos utilizar para dar soporte a la implementación de cada una de estas capas. Cada una de estas herramientas pueden aportar patrones de diseño específicos que son de gran utilidad para el tipo de aplicación que pretendemos desarrollar y por consiguiente del tipo de arquitectura a implementar.

3.4 Arquitectura SOA (Service Oriented Architecture)

Su objetivo es concebir las aplicaciones desde otro punto de vista, una aplicación orientada a servicios combina datos en tiempo real con otros sistemas capaces de fusionar los procesos de negocio. Las aplicaciones basadas en SOA utilizan tecnología totalmente estándar como es XML y Servicios Web para la mensajería. Estándares como SOAP , Web Services, Description Language (WSDL) y Business Process Execution Language (BPEL), estandarizan así la comparición de información, el modelo de integración de procesos y la cooperación entre aplicaciones.

Realizando aplicaciones orientadas a servicio se pueden conectar aplicaciones heterogéneas con el aumento de flexibilidad que supone, y un punto muy importante es que permite que las organizaciones interactúen cuando realmente lo requieran, sin necesidad de tener conexiones permanentes. Como una arquitectura SOA se basa en estándares, el tiempo de aprendizaje de utilización de las tecnologías sobre las que se apoya se reduce drásticamente.

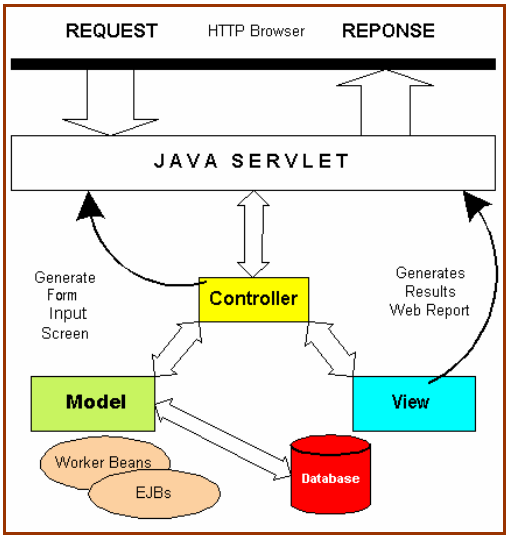
Una aplicación SOA la podemos dividir en tres capas. La capa de recepción de peticiones (servicios controladores: encargados de recibir las peticiones de los clientes y realizar las llamadas necesarias a otros servicios, en la secuencia adecuada, para devolver una respuesta), la capa de tareas (servicios de negocio: representan una tarea de negocio) la capa de lógica reutilizables (servicios de utilidad: representan tareas muy reutilizables como tareas de negocio, acceso a datos...).



3.5 Modelo - Vista - Controlador (MVC)

Se realiza una mención especial de este patrón por ser uno de los más famosos y extendido. Este patrón está asociado a 3 capas, podríamos decir que suele estar presente allí donde estén 3 capas, pero es más bien una distribución fina de la secuencia de ejecución entre que se produce un evento en la capa de presentación y este es atendido en forma completa. Sus partes son.

- **Vista**, el componente que recibió el estímulo (un botón en un formulario, un check box, etc.) y generó un evento que puede involucrar el estado de los otros controles del formulario. Pertenece a la capa de presentación y está enteramente en el cliente y suele estar presente en parte del middleware.
- **Modelo**, componente asociado a entidades del dominio. No los procesos de negocio pero sí las entidades. Respecto de la arquitectura en 3 capas, entonces, el modelo incluye parte de la capa de negocio (entidades, no procesos) y toda la capa de acceso a datos. Respecto de la arquitectura en 3 partes, el Modelo se despliega (deployment) sobre el back-end entero, y también parte en el middleware
- **Controlador**, el componente que falta: el que se asocia a los procesos de negocio (el Modelo sólo se intersecta con las entidades de negocio). El Controlador recibe el evento que la vista generó y moviliza procesos del negocio que terminan cambiando el estado en el Modelo. Estos procesos de negocio suelen estar alineados a los casos de uso de la aplicación. Respecto de la arquitectura en 3 capas, el Controlador está comúnmente en la capa de negocio aunque puede tener una participación menor en la capa de presentación (según cada canal cliente, la pieza que recibe elevento que genera la Vista). Respecto de las 3 partes, el controlador suele estar normalmente en el middleware aunque en menor tiene cierta presencia en el front-end.



Este patrón puede ser el punto de partida de muchas aplicaciones Web, pero en muchas ocasiones no resulta suficiente para la aplicación que necesitamos desarrollar. En muchas ocasiones hay que ir más allá, entrando en un diseño arquitectónico basado en n capas o m partes, con patrones de diseño diferentes para cada una de ellas.

También podemos encontrarnos en la situación en que necesitamos una arquitectura diferente para un tipo de aplicaciones específicas. Un ejemplo puede ser Rich Internet Applications (Aplicaciones Ricas de Internet). Esta surge como una combinación de las ventajas que ofrecen las aplicaciones Web y las aplicaciones tradicionales. Pretender evitar las constantes recargas de página, ya que desde el principio se carga toda la aplicación y sólo se produce comunicación con el servidor cuando se necesitan datos externos como datos de una Base de Datos o de otros ficheros externos. Dependiendo del tipo de aplicación puede ser una opción muy adecuada (multimedia...)

4 PATRONES DE DISEÑO

Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y rehusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias). Su objetivo es agrupar una colección de soluciones de diseño que son válidas en distintos contextos y que han sido aplicadas con éxito en otras ocasiones. Los patrones de diseño son soluciones de sentido común que deberían formar parte del conocimiento de un diseñador experto. Además facilitan la comunicación entre diseñadores, pues establecen un marco de referencia (terminología, justificación). Además, facilitan el aprendizaje al programador inexperto, pudiendo establecer parejas problema-solución.

4.1 Tipos de Patrones

Con la aparición del libro 'Design Patterns: Elements of Reusable Object Oriented Software' en el año 1994, escrito por los ahora conocidos Gang of Four (GoF, que en español es la pandilla de los cuatro) formada por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides, se recopilaron y documentaros 23 patrones de diseño aplicados usualmente por expertos diseñadores de software orientado a objetos. Ellos no son los inventores ni los únicos involucrados, pero fue a partir de la publicación de este libro cuando empezó a difundirse con más fuerza la idea de patrones de diseño. El grupo de GoF creó 3 grandes categorías:

- **Creacionales**: Comprenden aquellos patrones centrados en la forma de crear las clases y sus instancias, así como el uso que recibirán una vez creadas. Su objetivo es abstraer el proceso de instanciación y ocultar los detalles de cómo los objetos son creados o inicializados. Ejemplos son el patrón Factoría, Factoría abstracta o Singleton
- **Estructurales**: Son aquellos patrones centrados en la composición y estructura de las clases y en la herencia entre ellas. Describen como las clases y objetos pueden ser combinados para formar grandes estructuras y proporcionar nuevas funcionalidades. Estos objetos adicionados pueden ser incluso objetos simples u objetos compuestos. Ejemplos son el patrón Adaptador, Facade, Bridge, Composite.
- **Comportamiento**: Comprenden aquellos patrones centrados en la comunicación entre las distintas clases y objetos. Nos ayudan a definir la comunicación e iteración entre los objetos de un sistema. Su propósito es reducir el acoplamiento entre los objetos. Ejemplos de patrones de este tipo son el patrón Interpreter, Iterator, Observer, Visitor.

Existe una lista detallada de patrones según esta clasificación en el siguiente enlace: Patrón de diseño .

4.2 El Catalogo de Patrones J2EE (Core J2EE Patterns)

A partir de la aparición del J2EE, apareció todo un nuevo catálogo de patrones de diseño. Desde que J2EE es una arquitectura por si misma que involucra otras arquitecturas, como Servlets, JavaServer Pages, Enterprise JavaBeans ..., merece su propio conjunto de patrones específicos para diferentes aplicaciones empresariales.De acuerdo al libro 'J2EE PATTERNS Best Practices and Design Strategies', existen 5 capas en la arquitectura J2EE: Cliente, Presentación, Negocios, Integración, Recurso. El libro explica 15 patrones J2EE que están divididos en 3 de las capas: presentación, negocios e integración. Algunos son:

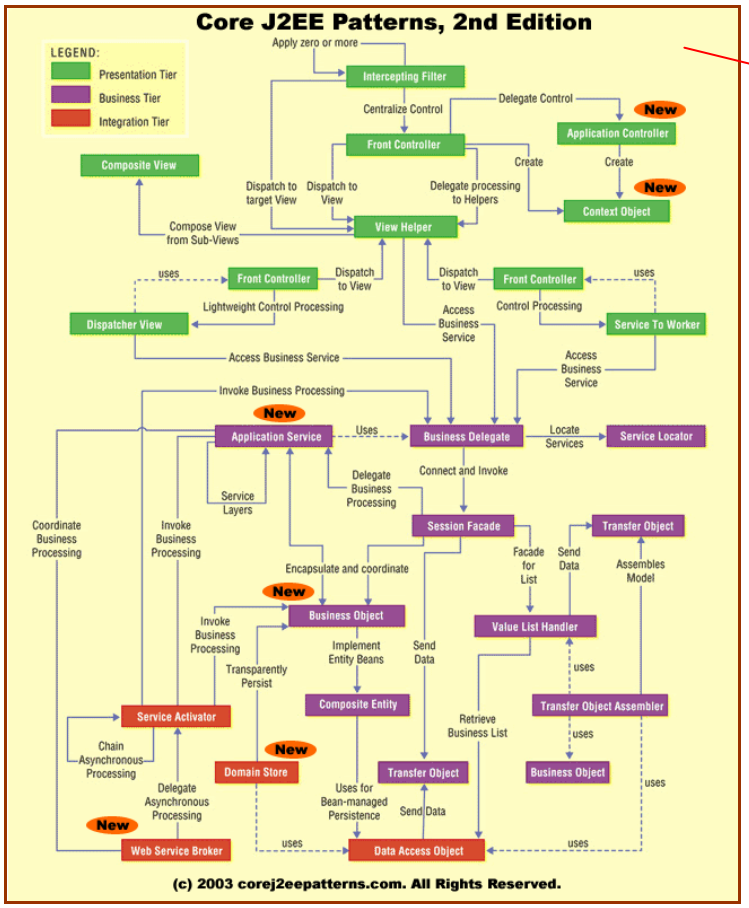
CAPA PRESENTACIÓN	
Decorating o intercepting Filter	Un objeto que está entre el cliente y los componentes Web. Este procesa las peticiones y respuestas.
Front Controller/ Front Component	Un objeto que acepta todos los requerimientos de un cliente y los direcciona a manejadores apropiados. El patrón Front Controller podría dividir la funcionalidad en 2 diferentes objetos: el Front Controller y el Dispatcher. En ese caso, El Front Controller acepta todos los requerimientos de un cliente y realiza la autenticación, y el Dispatcher direcciona los requerimientos a manejadores apropiada.
View Helper	Un objeto helper que encapsula la lógica de acceso a datos en beneficio de los componentes de la presentación. Por ejemplo, los JavaBeans pueden ser usados como patrón View Helper para las páginas JSP.
Composite view	Un objeto vista está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include o el action include es un patrón Composite View.

Service To Worker	Es como el patrón de diseño MVC con el Controlador actuando como Front Controller pero ,aquí el Dispatcher (el cual es parte del Front Controller) usa View Helpers a gran escala y ayuda en el manejo de la vista.
Dispatcher View	Es como el patrón de diseño MVC con el controlador actuando como Front Controller pero con un asunto importante: aquí el Dispatcher (el cual es parte del Front Controller) no usa View Helpers y realiza muy poco trabajo en el manejo de la vista. El manejo de la vista es manejado por los mismos componentes de la Vista.

CAPA NEGOCIOS	
Business Delegate	Un objeto que reside en la capa de presentación y en beneficio de los otros componentes de la capa de presentación llama a métodos remotos en los objetos de la capa de negocios.
Value Object/ Data Transfer Object/ Replicate Object	Un objeto serializable para la transferencia de datos sobre lar red.
Session Façade/ Session Entity Façade/ Distributed Façade	El uso de un bean de sesión como una fachada (facade) para encapsular la complejidad de las interacciones entre los objetos de negocio y participantes en un flujo de trabajo. El Session Façade maneja los objetos de negocio y proporciona un servicio de acceso uniforme a los clientes.
Aggregate Entity	Un bean entidad que es construido o es agregado a otros beans de entidad.
Value Object Assembler	Un objeto que reside en la capa de negocios y crea Value Objects cuando es requerido.
Value List Handler/ Page-by-Page Iterator/ Paged List	Es un objeto que maneja la ejecución de consultas SQL, caché y procesamiento del resultado. Usualmente implementado como beans de sesión.
Service Locator	Consiste en utilizar un objeto Service Locutor para abstraer toda la utilización JNDI y para ocultar las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y recreación de objetos EJB. Varios clientes pueden reutilizar el objeto Service Locutor para reducir la complejidad del código, proporcionando un punto de control.

CAPA INTEGRACIÓN	
Data Access Object Service Activator	Consiste en utilizar un objeto de acceso a datos para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.
Service Activator	Se utiliza para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.

El siguiente cuadro presenta el Catálogo de Patrones Principales de J2EE (Core J2EE Patterns):



Para tener una información más detallada sobre cualquiera de estos patrones se puede acceder:

Página Oficial de SUN.

Existe una versión en castellano disponible en:

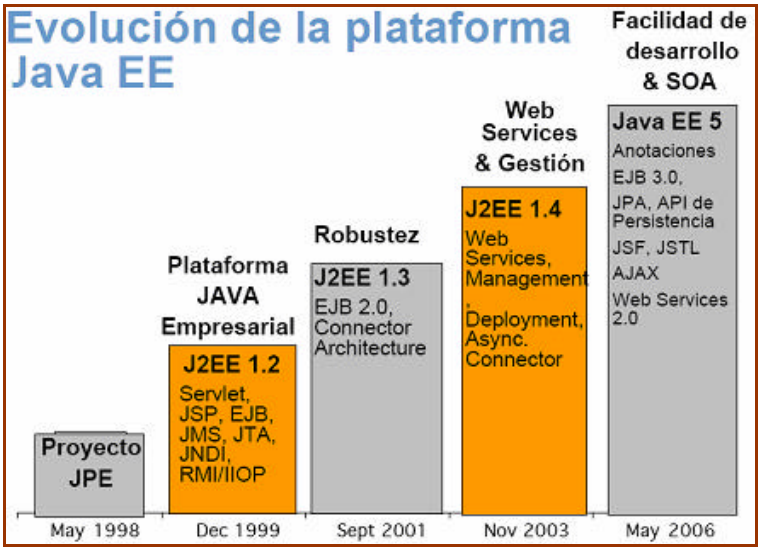
Capa de Presentación
Capas de Negocio e Integración

5 ESPECIFICACIONES

Centrándonos ya completamente en el mundo Java, según la definición que podemos encontrar en SUN:

> *'Java Platform, Enterprise Edition (Java EE) is the industry standard for developing portable, robust, scalable and secure server-side Java applications. Building on the solid foundation of the Java Platform, Standard Edition (Java SE), Java EE provides web services, component model, management, and communications APIs that make it the industry standard for implementing enterprise-class service-oriented architecture (SOA) and next-generation web applications.'*

Por tanto nos encontramos ante un estándar para el desarrollo de aplicaciones java portables, robustas, escalables y seguras. O sea, Java EE, provee un grupo de especificaciones diseñadas por Sun que permiten la creación de aplicaciones empresariales unificadas y compatibles para un determinado entorno de ejecución. Algunas de las especificaciones que contiene, dependiendo de la versión, serían: acceso a base de datos (JDBC), utilización de directorios distribuidos (JNDI), acceso a métodos remotos (RMI/CORBA), funciones de correo electrónico (JavaMail), aplicaciones Web (JSP, Servlets, EJB, Web Services...)...etc. Aquí es importante notar que Java EE es solo una especificación, esto permite que diversos productos sean diseñados alrededor de estas especificaciones algunos son Tomcat y Weblogic. La especificación más reciente de Sun es Java EE 5 , la cual esta conformada por: JSP 2.0 ,Servlet 2.4, EJB 2.1 y Connector 1.5 entre otros API's, los detalles se encuentran en aporta especificaciones alrededor de Web Services 2.0, Ajax, EJB 3.0, JSF, Api Persistencia, JRuby on Rails.... A continuación se presenta un cuadro que pretende mostrar la evolución de las especificaciones en función de las diferentes versiones que han ido apareciendo en el mercado:



Aunque varios productos Java están diseñados alrededor de estas especificaciones, no todos cumplen con el estándar completo, esto es, Tomcat solo emplea / cumple las especificaciones de JSP y Servlets, sin embargo, existen productos como Websphere y algunos otros "Java Application Servers" que son considerados "Fully J2EE Compliant" , en otras palabras, cumplen con todas las especificaciones definidas por Sun.

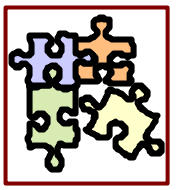
Sun ofrece una implementación de todas estas especificaciones llamada J2EE SDK, a pesar que el J2EE SDK es una implementación, esta es solo una muestra ofrecida por Sun para utilizar las funcionalidades de las especificaciones J2EE, aunque funcionaría en un Sistema de Producción su diseño es poco escalable además que su licencia lo prohíbe.

Podemos encontrar un listado completo de todas las especificaciones consideradas como finales en la Web de la Java Community Process (creado en 1998:proceso formalizado por la comunidad Java para definir especificaciones estándar):

Listado de Especificaciones Finales JSR

6 FRAMEWORK WEB

Una vez estudiados los diferentes patrones y el concepto de especificaciones aportadas por Java EE, es importante entrar a comentar el papel destacado que pueden tener los frameworks en el desarrollo de aplicaciones.Un framework es la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que (opcionalmente) pueden ser extendidas. En general, con el término framework, nos estamos refiriendo a una estructura software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta.



Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones. Un framework Web, por tanto, podemos definirlo como un conjunto de componentes (por ejemplo clases en java, descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas Web.

6.1 Tipos de framework Web

Existen varios tipos de frameworks Web: orientados a la interfaz de usuario, como Java Server Faces, orientados a aplicaciones de publicación de documentos, como Coocon, orientados a la parte de control de eventos, como Struts y algunos que incluyen varios elementos como Tapestry.La mayoría de frameworks Web se encargan de ofrecer una capa de controladores de acuerdo con el patrón MVC de Servlets y JSP, ofreciendo mecanismos para facilitar la integración con otras herramientas para la implementación de las capas de negocio y presentación.

6.2 Características

A continuación enunciamos una serie de características que podemos encontrar en prácticamente todos los frameworks existentes.

- **Simplificación:** Al usar el modelo MVC, la estructura interna de las aplicaciones se simplifica. La consecuencia es la reducción de los errores, más facilidad de testeo, más fácil de mantener… Usando un framework se reduce considerablemente el tiempo de desarrollo de un proyecto, siempre que éste justifique su uso.
- **Unificador:** Se establece un a estructura común para el desarrollo de aplicaciones, esto es básico cuando crece considerablemente el número de desarrolladores. Además se comparte la misma estructura de desarrollo en diferentes aplicaciones, esto hace que sea más fácil entender su funcionamiento evitando tener que el diseño arquitectónico de cada aplicación.
- **Integración:** Facilita la integración de aplicaciones que utilicen la misma arquitectura.
- **Abstracción de URLs y Sesiones:** No es necesario manipular directamente las URLs ni las sesiones, el framework ya se encarga de hacerlo.
- **Acceso a datos:** Incluyen las herramientas e interfaces necesarias para integrarse con herramientas de acceso a datos, en BBDD, XML, etc...
- **Controladores:** La mayoría de frameworks implementa una serie de controladores para gestionar eventos, como una introducción de datos mediante un formulario o el acceso a una página. Estos controladores suelen ser fácilmente adaptables a las necesidades de un proyecto concreto.
- **Autentificación y control de acceso:** Incluyen mecanismos para la identificación de usuarios mediante login y clave pudiendo restringir el acceso a determinas páginas a determinados usuarios.

6.3 Frameworks Java

En el momento de plantearse la construcción de una aplicación Web, una de las primeras dudas que se plantean gira entorno a la forma de implementarla. En este sentido, aspectos como la robustez, fácil escalabilidad, eficiencia, fácil mantenimiento, facilidad de programación, seguridad o precio se vuelven fundamentales.

La arquitectura J2EE surgió como un primer estándar básico de desarrollo, pero es evidente que, además, dependiendo del tipo y tamaño de la aplicación a realizar, se puede hacer necesario el uso de un framework o capa superior que asegure la calidad del proyecto facilitándonos su desarrollo y mantenimiento. Hoy en día, existe una gran variedad de productos en el mercado que implementan esta arquitectura.

Muchos de los problemas que se presentan en el desarrollo de aplicaciones Web no existen en las aplicaciones de escritorio y se deben, fundamentalmente, a que el protocolo HTTP, en el cual se basa la comunicación cliente-servidor, es un protocolo sin estado. Por tanto ha de realizarse un trabajo extra en el lado del servidor para poder mantener una conversación con el cliente durante toda la sesión. Entre estos problemas podemos destacar la validación y procesamiento de formularios, seguridad y control de acceso, internacionalización (adaptación de contenidos por país), separación entre diseñador y desarrollador, encapsulación y separación de funciones...

En general, muchos de los frameworks siguen una filosofía parecida aunque es posible clasificarlos en distintos grupos, en función de la capa sobre la que están especializados (presentación, lógica, integración). Esto es síntoma de que muchos de ellos aparecen sin añadir ninguna novedad a los frameworks ya existentes

7 ESTUDIO DE FRAMEWORKS

A continuación se presentarán algunos de los frameworks de código libre para Java existentes en el mercado. En particular se realizará una breve descripción, mostrando algunas de sus características más importantes. Se realizará una clasificación en función de la capa sobre la que están especializados.

Cabe resaltar que el hecho de que un framework sea un proyecto de la fundación Apache, probablemente la mejor organización de software Open Source que hay en el mundo, certifica su calidad.

7.1 Capa Presentación

STRUTS	URL Proyecto Struts

Es uno de los framework MVC más difundidos del mercado. Desarrollado dentro de la Apache Software Foundation, 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. Operativo bajo cualquier contenedor de Servlets.

Struts proporciona un controlador (Servlet) que se integra con una vista realizada con páginas JSP, incluyendo JSTL y Java Server Faces, entre otros. Este controlador evita la creación de Servlets y delega la operatoria en acciones creadas por el desarrollador, simplificando sobremanera el desarrollo de aplicaciones Web. Para facilitar las tareas comunes en la creación de esta capa existen una serie de tecnologías que se integran con Struts: Struts Taglib, JSTL, Tiles, Struts Validator, Struts Menú, CSS...

Aporta beneficios en aspectos como: lógica de navegación entre páginas, binding entre Java y el html (acceso rápido a las clases java desde las vistas), validación de entradas de cliente, internacionalización de la aplicación, independencia del motor de visualización, maquetización (página compuesta a porciones).Simplifica mucho el proceso de desarrollo de la capa de presentación, permitiendo dedicar más tiempo a codificar la lógica de negocio.

El resultado es una plataforma cooperativa, eficaz y apropiada tanto para desarrolladores independientes como equipos grandes de desarrollo. Al tratarse de uno de los primeros framewoks MVC que apareció en el mercado. Debido a ello conlleva una serie de errores de diseño que ha ido arrastrando. Otros frameworks han aprendido de el y ha evitado caer en ellos.

Recientemente ha aparecido la primera versión estable de Struts 2. Es el fruto de la unión de Struts con otro framework denominado WebWork de manera que se han aprovechado las ventajas de ambos frameworks y se ha rebautizado con el nombre de Struts 2. Esta versión incluye bastantes mejoras / cambios con respecto a la rama 1.0. Entre ellas cabria destacar: diseño mejorado, nuevos tags, mejora de objetos visuales (check box, botones...), soporte a Ajax, integración sencilla Spring, formularios POJO (abandona los ActionForm), acciones POJO, fácil

añadidura de plug-ins, mejor soporte al desarrollo (profiling, reporte de errores, debug), controladores fáciles de personalizar, fácil creación de nuevos Tags...	
Ventajas	Muchos años desde su aparición con desarrollos de envergadura completados con éxito. El objetivo del proceso de desarrollo es la calidad del producto. Muy popular, con mucho material disponible. Buenas prácticas conocidas. Mucha documentación. Curva de aprendizaje media. Permite crear aplicaciones Web de manera rápida y efectiva. Admite plug-ins para poder incrementar su funcionalidad. Se adapta a la incorporación de diferentes bibliotecas de tags. Struts2 Presenta un rediseño mejorado y mucha mejor funcionalidad. Código libre. Licencia Apache
Inconvenientes	No abstrae completamente del protocolo http. No esta diseñado para facilitar la creación de componentes propios. La vistas quedan atadas al dispositivo en el que se renderizan. No trata el concepto de transacción ni, por tanto, el de reutilización de transacciones, No define el nivel de lógica de negocio. Muy Orientado a la capa de presentación. Struts2 muy reciente (su última versión parece estable, pero versiones anteriores tenían errores graves)
Última Versión	6 de Julio de 2007 Struts 2.0.9
Google :Struts+Framework	340,000 resultados
Google :Struts2+Framework	150,000 resultados

TAPESTRY	URL Proyecto Tapestry
Tapestry complementa y se construye desde el Standard Java Serlvet API, funcionando, por tanto, en cualquier servidor contenedor de Servlets o contenedor de aplicaciones. Tapestry divide una aplicación Web en un conjunto de páginas, cada una compuesta de componentes. Esto le otorga una estructura consistente, permitiendo a Tapestry asumir responsabilidades clave como la construcción y envió URL, almacenamiento de estado persistente en el cliente o en el servidor, validación de entrada de usuario, localización / internacionalización, y reporte de excepciones. Desarrollar aplicaciones Tapestry implica crear plantillas HTML usando HTML plano, y combinando las plantillas con pequeños trozos de código Java usando un descriptor de archivos XML (opcional). Está claramente orientado a componentes, por tanto, se crean las aplicaciones en términos de objetos, y los métodos y propiedades de estos objetos. Ofrece un desarrollo realmente orientado a objetos a las aplicaciones Web Java. Desarrollar con Tapestry ofrece: transparencia en la construcción de vistas, binding entre Java y HTML, manejo de eventos, orientación a componentes, validación de entradas, internacionalización, soporte a XHR/ Ajax/ DHTML. soporte Ajax …	
Ventajas	Permite desarrollo componentes. Los plantillas se codifican en HTML estándar. No hay que aprender cosas nuevas. Curva de aprendizaje relativamente sencilla. Poca codificación Java. Separación completa entre lógica y presentación Código libre. Licencia Apache
Inconvenientes	Poco utilizado y aún poco extendido. En pleno proceso de expansión Comunidad de desarrolló pequeña. En Expansión. Escasa documentación. Pocos libros editados. Requiere configurar tres archivos para cada página a crear.
Última Versión	20 Septiembre 2007 Tapestry 4.1
Google: Tapestry+Framework	290,000 resultados

JAVA SERVER FACES	URL Proyecto Java Server Faces
JSF es un framework para construir interfaces de usuario en aplicaciones Web. Sabemos que la construcción de los interfaces de usuario suelen ser la parte más costosa del esfuerzo de desarrollo, sobre todo por la dificultad de mantenimiento. En especial cuando se mezcla en un mismo archivo JSP el interfaz de usuario, las reglas de validación y el acceso a la base de datos. JSF nos ofrece un marco de trabajo que facilita el desarrollo de aplicaciones, separando las diferentes capas de una arquitectura: presentación, reglas y entidades de negocio. En este sentido toma una orientación semejante a Struts. JSF fue creado dentro del Java Community Process de SUN, en el que han participado líderes de la industria como Oracle, BEA, IBM, etc. Es el framework oficial de SUN para el desarrollo de aplicaciones. Uno de los líderes, Craig McClanahan, es el creador del popular framework Struts. Además, como un desarrollo adicional al JCP, encontramos dentro del proyecto Yakarta el framework MyFaces. A continuación presento algunos de los puntos por los que JSF parece una tecnología muy interesante: Un modelo de trabajo basado en componentes UI (user interface), definidos por medio de etiquetas y XML. Incluye la capa de control, definida de forma declarativa en archivos XML. Lo que implica control de eventos y errores. Hay una serie de especificaciones que definen JSF(JSR 127, JSR 252,JSR 276). Trata la vista (el interfaz de usuario) de una forma algo diferente a lo que estamos acostumbrados en aplicaciones Web. Sería más similar	

al estilo de Swing, donde la programación del interfaz se hacer a través de componentes y basada en eventos (se pulsa un botón, cambia el valor de un campo,...).
Validación en cliente y en servidor.
JSF es muy flexible. Por ejemplo nos permite crear nuestros propios componentes.
Lógica de navegación entre páginas.
Binding entre la vista y los beans de negocio.
Manejo de eventos. Internacionalización. Independencia del dispositivo de presentación. Soporte Ajax.

JSF no puede competir en madurez con Struts (en este punto Struts es claro ganador), pero si puede ser una opción muy recomendable para nuevos desarrollos, sobre todo si todavía no tienen experiencia con Struts.

Ventajas	SUN esta detrás. Implementaciones de distintos fabricantes (MyFaces)... Al ser moderno ha aprendido de otros. Validación de gran alcance. Dificultad Media. Código libre.
Inconvenientes	Falta algo de documentación.
Ultima Versión	11 Mayo 2006 –Version 1.2 latest release.
Google : JSF + Framework	473.000 Resultados

MyFACES	URL Proyecto MyFaces

Es un desarrollo de código abierto creado por la fundación Apache de la especificación JSF (JSR- 127). Básicamente, se trata de un marco de desarrollo de aplicaciones Web siguiendo el patrón MVC, pero a diferencia de otros desarrollos que persiguen el mismo objetivo, como Struts, MyFaces tiene una fuerte orientación hacia considerar los componentes de la aplicación, botones, cajas de texto, etc. , como objetos de primera categoría, todo se construye alrededor de ellos, haciendo de esta forma que el desarrollo de las aplicaciones Web sea parecido al que podríamos encontrar en un programa creado para escritorio. Sus objetivos son parecidos a los de Tapestry, Wicket.

MyFaces es un desarrollo de la especificación JSF, lo que implica que para usarlo correctamente, antes habrá que haber estudiado, y muy importante, entendido, qué es y para qué vale la especificación JSF.

Se caracteriza por incorporar un gran número de componentes adicionales, además de los exigidos por la especificación JSF Los programadores de Jakarta han hecho un esfuerzo creando un sin fin de componentes adicionales que son de gran utilidad en las aplicaciones Web, menús, listas, etc. Estos componentes se agrupan en una librería que recibe en nombre de tomahawk. Estos componentes se podrían usar en cualquier distribución alternativa de JSF ya que están desarrollados siguiendo los estándares de construcción de componentes JSF. Además, no hace demasiado, Oracle decidió donar a la fundación Apache, la librería de componentes JSF que había sido creada por ellos ADFFaces, por lo que el numero de componentes sigue creciendo.

Otro punto donde MyFaces ha completado a la especificación JSF original, es en el tema de las validaciones. JSF sólo obliga a que las validaciones se hagan en el servidor, y dejando a un lado si esto es lo correcto o no, MyFaces da la posibilidad de que estas validaciones se realicen también en el cliente.

Si como tecnología de vista no nos decantamos por JSP, es posible utilizar librerías JSP tradicionales, por ejemplo, JSTL, pero es importante intentar usar solo componentes JSF, para que la aplicación quede más elegante , y no haya una mezcla de tecnologías, que repercutan en un deficiente mantenimiento.

Ventajas	Todas las de JSF. Más y mejorados Componentes. Muy extendido y usado. Mucha Documentación. Curva de aprendizaje baja/media. Mejorar en la implementación de validaciones Código libre. Licencia Apache.
Inconvenientes	Se trata de la implementación de un tercero. Los de JSF
Ultima Versión	2 Junio 2007 - MyFaces Tomahawk 1.1.6 Released
Google : MyFaces+Framework	141.000 Resultados

Spring MVC	URL Proyecto Spring MVC

Es muy flexible, ya que implementa toda su estructura mediante interfaces no como Struts 1.0 que hasta ahora obligaba a heredar de clases concretas (tanto en Acciones como en Formularios). Además, todas las partes del framework son configurables vía plug-in en la interfaz, aunque también provee clases concretas como opción de implementación. Es flexible en la selección del lenguaje para integrarlo en la View. No obliga a usar JSP, permite usar XLST o Velocity o realizar nuestra propia implementación de lenguaje.

Los controladores de Spring MVC se configuran mediante IoC como los demás objetos, lo cual los hace fácilmente testeables e integrables con otros objetos que estén en el contexto de Spring, y por tanto sean manejables por éste.

La capa Web de Spring es una pequeña parte en lo alto de la capa de negocio de Spring, lo cual parece una buena práctica. Struts y otros frameworks Web dejan como libre elección la implementación de los objetos de negocio, mientras que Spring ofrece un framework para todas las capas de la aplicación. Spring tiene una interfaz bien definida para la capa de negocio. Además ofrece soporte Ajax.

Al igual que Struts 2.0, provee de unos objetos llamados interceptores, que actúan a modo de controladores y que en función de su configuración sirven para tratar múltiples tipos de peticiones de cliente. Es una de las alternativas a Struts que ha incorporado una lógica de diseño más sencilla y que cuenta con todo el abanico de librerías de Spring. No obstante sigue también la misma filosofía que Struts 2 pretendiendo, ambos. dar solución al mismo tipo de problemas desde un enfoque parecido

Ventajas	Ofrece una división limpia entre Controllers, Models (JavaBeans) y Views. Es un framework completo para todas las capas. Código libre. Licencia Apache
Inconvenientes	Mucha configuración XML No demasiado extendido. Curva de aprendizaje Media/alta
Ultima Versión	Spring Framework 2.0.7 release
Google : Spring+MVC+Framework	108,000 Resultados

TURBINE	URL Proyecto Turbine

Es uno de los primeros frameworks MVC, y permite el desarrollo de aplicaciones Web, orientadas a servicios, seguras en poco tiempo. Está orientada a desarrolladores Java con experiencia. Una ventaja es que las herramientas provistas por Turbine es que sus partes pueden ser separadas, participando en el desarrollo de nuevos proyectos. Por tanto, permite la reutilización. Algunas de las funcionalidades ofrecidas son la inclusión de sistemas de gestión de la seguridad, servicios de scheduling, sistemas de validación de formularios basados en XML y un servicio XML-RPC para el desarrollo de WebServices. Por otro lado, se trata de un framework bastante pesado, con una serie de servicios añadidos, que impactaban en el rendimiento. Lo más importante, en mi opinión, es que a la sombra de Turbine se desarrollaron una serie de subproyectos interesantes. Por ejemplo, Velocity se creó como su lenguaje de plantillas, Maven como herramienta de gestión, Torque como framework de acceso a datos,…

Se recomienda que las aplicaciones construidas con Turbine sigan la arquitectura definida por el patrón de diseño MVC. Esto facilita su programación y mantenimiento.

La base de Turbine es totalmente independiente de la tecnología usada en la capa de presentación. Tanto Java Server Pages (JSP) como Velocity están soportados por Turbine. Para desarrolladores familiarizados con JSP, o con JSP Tag Libraries, Turbine ofrece soporte para el estándar de Sun. Velocity es la tecnología preferida para la implementación de la capa de presentación, para la mayoría de programadores que utilizan este framework.

El desarrollo de este framework se realiza en un entorno de libre participación y cuenta con la licencia de la Fundación de Apache.

El 16 de Mayo de 2007 - Turbine pasa a formar parte de los proyectos de alto nivel de la Apache Software Foundation (ASF). Con esto se separa de Jakarta para pasar a trabajar de forma autónoma.

Ventajas	Genéricas de los framewoks. Código libre. Licencia Apache.
Inconvenientes	Poco utilizado y aún poco extendido. En pleno proceso de expansión. Comunidad de desarrolló pequeña. En Expansión. Pesado y de mejorable rendimiento. Curva de aprendizaje media / alta. Cierta complejidad de uso y mantenimiento.
Última Versión	3. Octubre 2005 - Turbine 2.3.2 released
Google : Turbine+Framework	319,000 resultados

COCOON	URL Proyecto Cocoon

Apache Cocoon es un sistema de publicación Web, basado en XML/XSL. Cuenta con desarrollo total en Java por lo cual se puede ejecutar desde cualquier servidor que pueda contener Servlets; y al ser un Servlet cuenta con las ventajas de éstos, es decir, se ejecutan como threads de forma simultánea en el mismo contexto.

Es bastante configurable y personalizable. Además adopta características para escribir páginas de servidor en XML (XSPs -eXtensible Server Pages). Permite diferenciar el procesamiento del documento para tenerlo en distintos formatos, dependiendo del tipo de software que hace la petición y cuenta con un sistema de caché para tener un mejor rendimiento. Un elemento adicional y clave para tener en cuenta es que es un producto gratuito.

Existen dos implementaciones de Cocoon. La primera (Cocoon1) nació como solución a las aplicaciones Web que necesitaban presentas contenidos dinámicos. Trabajaba sobre DOM (*Document Object Model*) para poder pasar los documentos XML entre componentes. El problema es que el trabajo con árboles DOM se vuelve ineficiente ya que el procesamiento de un árbol consume mucha más memoria que el documento XML original. La segunda (Cocoon2) está construida sobre el API SAX que es mucho más eficaz cuando se trata de manipular documentos XML. Se puede considerar Cocoon 1 obsoleto.

Con las XSP y Cocoon se puede tener acceso a una base de datos de cualquier tipo, con lo cual se puede tener la persistencia de una aplicación en un sistema manejador de bases de datos y aprovechar las ventajas tanto del manejador como de las XSP.	
Ventajas	Separación completa entre lógica y presentación Permite modificar el comportamiento de la aplicación sin conocer el lenguaje de implementación. Enfoque diferente e innovador. Código libre. Licencia Apache
Inconvenientes	Requiere conocimiento avanzados de XML, hojas de estilo XSL. Comunidad relativamente pequeña. Curva de aprendizaje elevada. Mantenimiento costoso. La tendencia no sigue los conceptos en que se basa (XML, XSL…). El procesado de XML requiere bastante capacidad de proceso.
Ultima Versión	21 de Diciembre de 2006 Versión 2.1.10
Google :Cocoon+Framework	266,000 resultados

Velocity	URL Proyecto Velocity
Es un framework de Apache (VWF) que permite desarrollar pequeños y medianos proyectos Web de manera rápida. Está basado en un conocido motor de plantillas (Jakarta Velocity). Contiene, además, un módulo de plug-ins por lo que es fácilmente ampliable. Permite a los diseñadores de páginas hacer referencia a métodos definidos dentro del código Java (binding). Velocity es una alternativa a Java Server Pages y en realidad es un lenguaje de plantillas y scripting en la página.	
Ultima Versión	6 Marzo 2007 Velocity 1.5
Google: Velocity + Framework	19,100 resultados

7.2 Capa Lógica de Negocio

SPRING	URL Proyecto Spring
El Spring Framework (también conocido simplemente como Spring) es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. Da soluciones a varias de las capas de la arquitectura Web (presentación, lógica de negocio, integración...). También hay una versión para la plataforma .NET, Spring.net.A pesar de que Spring Framework no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java al considerársele una alternativa y sustituto del modelo de Enterprise JavaBean, aunque también pueden convivir. Diseñado en módulos, con funcionalidades específicas y consistentes con otros módulos, te facilita el desarrollo de funcionalidades específicas y hace que la curva de aprendizaje sea favorable para el desarrollador. Además existen soluciones muy bien documentadas y fáciles de usar, prácticas comunes en la industria. Es un framework Java/J2EE que ofrece, entre otras cosas, contenedor ligero, soporte de AOP, abstracciones JDBC, meta datos a nivel de código fuente y framework MVC para Web. Entre los aspectos más interesantes es la posibilidad de aplicar transacciones declarativas con JTA (u otro tipo de transacciones) a clases planas Java (POJOs). Esto nos permite utilizar transacciones en sistemas ligeros web basados en Tomcat, por ejemplo. Tecnológicamente, incorpora dos conceptos que , aunque no son nuevos, se han puesto de moda al ser usados por algunos frameworks. IoC (Inversion of Control) y DI (Dependency Injection) que es un concepto más avanzado que IoC. Dependency injection generalmente implica que existe un contenedor de objetos y este último es el encargado de la creación de beans (de ahí el nombre de fábrica de beans) y el ensamblado de las dependencias de todos los beans que maneja (IoC). Además el contenedor es el responsable de todo el ciclo de vida de los beans. Otro aspecto muy interesante son sus plantillas para Hibernate, permitiendo realizar el acceso a datos con una única línea de código, lo que simplifica enormemente la creación de lógica de persistencia. Lo mismo podríamos decirlo para los plantillas de Jdbc que también ofrece.	
Ventajas	La DI aporta ventajas muy interesantes como las plantillas para los frameworks ORM u otras plantillas para otros servicios. Buen grado de desacoplamiento. Orientado a interfaces. Soporta la persistencia (mediante anotaciones o XML).Admite proveedores externos para esta capa. Soporta JDBC. Se puede ejecutar dentro de un Container Web o fuera de él en una aplicación Swing normal y corriente. Curva de Aprendizaje Media y mucho soporte para mantenimiento Herramientas adicionales como Spring IDE crea los grafos de beans. Código libre. Licencia Apache
Inconvenientes	Configuración de Spring es un poco costosa (mucho XML inherente al tipo de controlador) Perdida de las ventajas del tipado fuerte, ya que al inyectar objetos los fallos sólo pueden detectarse en tiempo de ejecución. Su container no es ligero (pensado para aplicaciones adecuadas. No de tiempo real o en una aplicación para un móvil...)
Ultima Versión	Spring Framework 2.0.7 release
Google :Spring+Framework	4.500.000 resultados

EJB 3.0	URL Proyecto EJB
El modelo de programación propuesto por la versión 2.1 de EJB conllevaba una serie de inconvenientes que limitaron mucho el uso de esta especificación y conllevó la aparición de soluciones open source que suplían las carencias que presentaba EJB 2.1. Parece que las soluciones open source que más han marcado el desarrollo empresarial dentro de la plataforma Java han sido Hibernate y Spring Framework, y en la nueva versión de Java Enterprise Edition se han incorporado muchas de las características de estos frameworks para procurar a los desarrolladores una plataforma de desarrollo bastante más sencilla que su predecesora versión 1.4 Es una arquitectura de componentes para el desarrollo y despliegue de aplicaciones empresariales orientadas a objetos y distribuidas. Realizado por SUN y que sigue la especificación JSR 220 FR es el único estándar Java que se ocupa de la lógica de negocio del lado del servidor, y esto es fundamental para J2EE. El comité de especificación de EJB reconoce que EJB se ha quedo corto en alcanzar algunas de sus ambiciosas metas, y necesita una modernización. Esta modernización está muy enfocada en la facilidad de uso, principalmente mediante la simplificación de los requerimientos de los implementadores de beans. Sin embargo, el comité de especificación también ha identificado un número de mejoras funcionales críticas que facilitan el uso y la eliminación de ciertos anti-patrones J2EE. Algunas de las mejoras técnicas que incorpora son IoC (Inversion of Control), DI (Dependency Injection), realmente orientado a objetos, utilización de anotaciones (no necesario implementar XML para acceso a datos, bastaría con especificar que la clase es una entidad, cual es el campo identificador y poco más), el mapeo relacional pasa a ser parte de la especificación.	
Ventajas	Se rige por una especificación estándar e implementada por los App Servers J2EE. Alto grado de madurez. Respaldad por Sun Microsystems. Soporta la persistencia (mediante anotaciones o XML). Curva de aprendizaje Media. Mejorada respecto a versiones anteriores.
Inconvenientes	No soporta JDBC. Container pesado. Cierta falta de flexibilidad. Revisión bastante reciente. Curva de aprendizaje Media.
Ultima Versión	EJB 3.0
Google :EJB+Framework	520.000 resultados

HIVEMIND	URL Proyecto HiveMind
HiveMind propone una arquitectura de servicios totalmente declarativa, eliminando dependencias en el código. El propósito del producto es usar el patrón Dependency Injection para eliminar de nuestro código esas tareas tediosas como puede ser la invocación al servicio de log… HiveMind consiste en un registro de servicios y configuraciones. La zona superior, es la aplicación del usuario, responsable de crear el registro y obtener servicios de él. Esta aplicación puede ser una aplicación Web basada en Servlets, una aplicación basada en consola, Swing... HiveMind puede emplearse con cualquier tipo de aplicación. Cada servicio es un interfaz combinado con un POJO que la implementa. Estos servicios son los contenedores de lógica de negocio, muy similares a los beans de sesión sin estado. HiveMind se encargara de crear instancias de los servicios. Las configuraciones de HiveMind son los contenedores de XML; una especie de descriptores de despliegue genéricos para cualquier clase de datos necesarios para la aplicación. HiveMind también incluye una herramienta de documentación, HiveDoc que crea una representación en hipertexto de los servicios y configuraciones que forman un registro, permitiendo ver de un modo simple cómo está estructurado un registro. Es, después de Spring, una de las opciones más interesantes con una filosofía paralela. De la mano de Apache es un completo contenedor IoC. Se han visto sin embargo tres grandes áreas en donde Spring es superior: conjunto de librerías, apoyo de la comunidad a todos los niveles y elegancia de uso.	
Ventajas	
Inconvenientes	Muy poco utilizado Poca documentación Muy pocos desarrollos
Ultima Versión	V 1.1
Google :Hivemind+Framework	56.200 resultados

AVALON/EXCALIBUR	URL Proyecto Avalon \ Excalibur
Descripción	
Es uno de los contenedores IoC más antiguos. Sin embargo, lejos de ser una ventaja, esto se nota mucho en su diseño, mucho menos flexible y elegante que Spring. Su comunidad es realmente pequeña y no se han encontrado muchas referencias a usos en aplicaciones reales.	
Ventajas	Código libre. Licencia Apache
Inconvenientes	Antiguo y un poco desfasado. Poca documentación y poco usado.
Ultima Versión	5 de Julio 2007 Excalibur 2.2.3
Google Excalibur+Framework	36.300 resultados

SEAM 2	URL Proyecto Seam

Producto creado por Gavin King el creador de Hibernate y que busca la creación rápida de aplicaciones web java. Para ello utiliza una herramienta llamada seam-gen que te genera a partir de tablas en una base de datos una aplicación CRUD (create, read, update, delete) totalmente funcional con tecnologías estándar de JEE como JSF para la vista, EJB3 para la lógica del negocio y JPA para la persistencia y tecnologías de JBoss como Hibernate Validation o JBoss jBPM para definir el flujo de una aplicación.
En esta nueva versión encontrarás las siguientes características:

- Seam WS que permite que un componente Seam sea un servicio web
- Creación de componetes Seam con Groovy
- Independencia de JSF, por lo que en teoría puedes usar cualquier otro framework web para tu capa de vista
- Soporte para Google Web Toolkit
- Integración con Hibernate Search
- Soporte para transacciones no JTA
- Integración de Ajax4JSF

7.3 Capa de Integración

JDBC	URL JDBC

JDBC (Java Database Connectivity) es una especificación (incluida dentro de las especificaciones Java EE) de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea, independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice. La aplicación Java que se desarrolle debe tener acceso a un driver JDBC adecuado. Este driver es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.
Los drivers JDBC, al estar escritos en Java son automáticamente instalables, portables y seguros.

Ventajas	Muy utilizado y extendido. Utilización directa sin necesidad de aprender o configurar otros entornos.
Inconvenientes	Código específico de Base de Datos. Baja Portabilidad. Alto acoplamiento. Acceso directo a conexiones y transacciones. Transacciones a un único DataSource El desarrollador Java debe saber también otro lenguaje (SQL). El código se hace repetitivo: Inserts, Selects, Updates, Deletes para cada objeto del modelo.
Ultima Versión	11 Diciembre 2006 JDBC 4.0 API
Google :JDBC+Framework	489.000 resultados

HIBERNATE	URL Proyecto Hibernate

Es una herramienta mapeador objeto-relacional ORM completa que ha conseguido en un tiempo record una excelente reputación en la comunidad de desarrollo posicionándose claramente como el producto de código libre líder en este campo gracias a sus prestaciones, buena documentación y estabilidad. Es valorado por muchos incluso como solución superior a productos comerciales dentro de su enfoque, siendo una muestra clara de su reputación y soporte la reciente integración dentro del grupo JBoss que seguramente generará iniciativas muy interesantes para el uso de Hibernate dentro de este servidor de aplicaciones.

Para usar Hibernate se tienen que crear los objetos de negocio y sus relaciones. Después se deberá crear una serie de archivos xml que indican a la herramienta qué objeto se guarda en qué tabla, indicando la relación entre propiedades del objeto y columnas de la tabla. De esta forma ya tenemos mapeado el objeto con la base de datos. Finalmente queda crear otro archivo xml para configurar la conexión que ha de usar para el acceso a los datos. Este proceso es simple y el resultado es un código más simple en el que se evita tener que usar sentencias SQL.

Uno de los puntos en los que hay que tener cuidado es en el hecho de que Hibernate guarda los datos de los objetos en la sesión , y hasta que no se le se le indica a Hibernate, los datos no están realmente persistidos. El hecho de que la sesión se interponga entre los datos y la base de datos obliga a tener mucho cuidado en como se gestionan las sesiones. Puesto que una sesión debe pertenecer a un único Thread y un único Thread no debería de tener más de una sesión porque en caso contrario no se puede asegurar la integridad de los datos

Ventajas	El mapeo objeto-relacional evita la creación de consultas SQL . Evitará código repetitivo dentro de la clases bean (set, get...). Permite recuperar el grafo de objetos y sus relaciones (Ingeniería Inversa). En la mayor parte de los casos (excepto los especiales) no es necesaria infraestructura de gestión jdbc: abrir y cerrar recursos, tipos especiales del estilo boolean, blob, timestamp, date... Plugins para eclipse e integrado en Spring y en otras herramientas. Es la tecnología ObjectRelationalMapping muy usada. Comunidad muy activa con mucha documentación. Código libre.
Inconvenientes	Curva media de aprendizaje. No es una tecnología estándar aunque, pese a ello parece, un punto superior al resto . No hay disponibles plugins para muchas herramientas de desarrollo (Websphere de IBM, Eclipse lo tienen).
Ultima Versión	31 de Julio de 2007 Hibernate 3.2.5
Google :Hibernate+Framework	1.200.000 resultados

IBATIS	URL Proyecto Ibatis
Es un framework de código abierto basado en capas, desarrollado por Apache Software Foundation, orientado a la gestión de persistencia que facilita el diseño de esta capa (se sitúa entre la lógica de Negocio y la capa de la Base de Datos) utilizada en las aplicaciones Java para acceder al repositorio de datos. Permite que el desarrollador se olvide de la implementación del acceso a datos, únicamente se debe preocupar por realizar una correcta configuración.No es un ORM puro(no orientado a objetos).No es totalmente transparente ya que asocia objetos de modelo (JavaBeans) con sentencias SQL o procedimientos almacenados mediante ficheros descriptores XML, simplificando la utilización de bases de datos.Dividiremos la capa de Persistencia en dos subcapas complementarias pero independientes, por lo que es posible implementar únicamente una de las dos: **1. Capa de Abstracción** - DAO, que implementa la capa de acceso a los datos. Se ocupa de la comunicación con la propia Base de Datos utilizando un driver específico para la misma. Es el interfaz con la capa de la lógica de negocio, haciendo las veces de 'facade' entre la aplicación y la persistencia. Se implementa mediante el patrón Data Access Object (DAO). **2. Capa de Persistencia** - SQL MAPS, implementa la capa de persistencia. Actúa de interfaz con el gestor de Base de Datos ocupándose de si gestión mediante una Api, que en Java es JDBC. Proporciona un modo simple y flexible de mapear los datos con los objetos Java. De este modo, se tiene toda la potencia de SQL sin una línea de código JDBC. SQL Maps no sólo reduce considerablemente la cantidad de código necesario para acceder a una base de datos relacional sino que también ofrece una separación entre la capa de negocio y la de acceso a Datos. Este framework mapea la clases a sentencias SQL usando un descriptor XML muy simple.	
Ventajas	iBATIS DAO facilita el manejo de la capa de persistencia mediante la utilización de interfaces. iBATIS SQL MAPS facilita la asignación de datos entre el modelo de datos de nuestra base de datos y el modelo de clases de nuestra aplicación. Simplicidad: fácil de aprender a utilizar y mantener. Programación declarativa (configurando ficheros XML). Separación entre SQL y el lenguaje de programación de la aplicación (Java/.NET). Portabilidad: entre Java y .NET; entre diferentes proveedores de bases de datos. Curva de aprendizaje rápida y pocos conocimientos previos (Altos de SQL) Alto rendimiento y optimización Se trata de código abierto, es decir código Abierto.
Inconvenientes	No es ORM. Las bases de datos que gestiona iBATIS deben ser exclusivamente relacionales. Codificación específica según la base de datos. Baja Portabilidad. Alto Acoplamiento. No es totalmente transparente (hay que programar SQL).
Ultima Versión	1 de Diciembre de 2006: iBATIS 2.3.0
Google :Ibatis+Framework	237.000 resultados

CASTOR	URL Proyecto Castor
Castor es un framework de persistencia ORM que mapea objetos JAVA a través del marshalling. El marshalling consiste en utilizar una serie de ClassDescriptors y FieldDescriptors para describir como un objeto debe ser marshalled y unmarshalled desde o hacia un XML. Para explicar un poco mejor los términos "marshal" y "unmarshal", decir que solo consiste en el hecho de convertir un stream (sequence of bytes) de datos en o desde un objeto. "Marshalling" consiste en convertir un Object a un stream, y "unmarshalling" de stream a Object. Para todo ello es necesario definir un archivo mapping.xml el cual contiene las definiciones necesarias para el mapeo.	
Ventajas	Curva de aprendizaje relativamente baja Herramienta tipo ORM.
Inconvenientes	Poco usado con falta de documentación Pocos ejemplos de uso.
Ultima Versión	28 de Junio de 2007 Castor 1.1.2.1
Google :Castor+Framework	329.000resultados

TORQUE	URL Proyecto Torque
Descripción	
Torque incorpora herramientas que permiten la gestión de la persistencia creando un modelo físico de datos adaptado a diferentes SGBD gracias al empleo de XML. Además permite evitar la codificación directa de sqls, así como abstraer al programador de la problemática vinculada con la apertura y cierre de conexiones a BBDD. Es una herramienta muy potente que se debe usar con cuidado, puesto que si se usa de manera inadecuada puede evitar los inconvenientes del acceso a BBDD a costa de introducir nuevos problemas.	
Ventajas	Evita el uso directo de Sql. Curva de aprendizaje relativamente baja. Control de conexiones.
Inconvenientes	El framework te crea demasiadas clases. Demasiados acceso a la bd, sobretodo cuando son consultas que involucran varias tablas. Poco usado con poca documentación
Ultima Versión	27 de Febrero de 2007 Torque 3.3
Google :Torque+Framework	256.000resultados

7.4 Capa de Seguridad

ACEGI	URL Proyecto Acegi
Acegi Security System proporciona la funcionalidad necesaria para adoptar mecanismos de seguridad en aplicaciones Java utilizando características de Programación Orientada a Aspectos (POA), de forma transparente para el desarrollador, sin necesidad de desarrollar código, utilizando para ello el soporte prestado por el framework Spring, pero siendo posible utilizarlo en aplicaciones no desarrolladas con Spring. ACEGI es un framework de seguridad open source, muy configurable que permite reutilizar y portar componentes de seguridad y que la lógica de negocio libre de código de seguridad. Proporciona la funcionalidad necesaria para adoptar mecanismos de seguridad en aplicaciones Java, especialmente en términos de autorización y autenticación, utilizando características de programación orientada a aspectos, de forma transparente para el desarrollador, sin necesidad de desarrollar código, utilizando para ello el soporte prestado por el framework Spring, pero siendo posible utilizarlo en aplicaciones no desarrolladas con Spring. Proporciona cuatro opciones principales de seguridad : 1. Listas de control de acceso (ACL) web basadas en esquemas URL 2. Protección de métodos y clases Java usando AOP 3. Single sign-on (SSO) 4. Seguridad proporcionada por el contenedor Web	
Ventajas	Proporciona mecanismos de seguridad transparentes
Inconvenientes	Muy Orientado a Spring Curva de aprendizaje media/alta
Ultima Versión	Mayo 2006 Acegi 1.0.0
Google :Acegi+Framework	971.000 resultados

HDIV	URL Proyecto HDIV
Para solventar estas limitaciones de Struts y eliminar las vulnerabilidades típicas de las aplicaciones web, surge el proyecto open-source HDIV (HTTP Data Integrity Validator) concebido como una ampliación de seguridad para Struts y Spring MVC. Se ha publicado la nueva versión 2.0 del framework web de Seguridad (HDIV), dirigido a solucionar las vulnerabilidades web de nivel aplicación (parameter tampering, SQL Injection, XSS), con soporte para Spring MVC y JSTL 1.1. Además esta nueva versión es compatible con las versiones Struts 1.3.8 y Struts 2.0.9.Las funcionalidades de seguridad añadidas a los frameworks Struts 1.x, Struts 2.x y Spring MVC son :: • Integridad: HDIV garantiza la integridad (la no modificación) de todos los datos generados en el servidor que no pueden ser modificados por la parte cliente (links, campos ocultos, listas seleccionables, valores de radio, páginas destino, cookies, etc.). Gracias a esta propiedad se eliminan todas las vulnerabilidades basadas en la técnica parameter tampering. • Validación de datos editables: HDIV elimina gran parte del riesgo originado por ataques de tipo Cross-site scripting (XSS) y SQL Injection mediante validaciones genéricas de datos editables (text y textarea). • Confidencialidad: HDIV garantiza la confidencialidad de todos los datos generados en el servidor. Habitualmente muchos de los datos enviados al cliente aportan información clave para los posibles atacantes como identificadores de registros de Bases de Datos, nombre de columnas o tabla, nombres de directorios web, etc.	
Ultima Versión	HDIV 2.0

7.5 Pruebas Unitarias

JUNIT	URL JUnit
En los últimos años se han desarrollado un conjunto de herramientas que facilitan la elaboración de pruebas unitarias en diferentes lenguajes. Dicho conjunto se denomina XUnit. De entre dicho conjunto, JUnit es la herramienta opensource formada por un conjunto de clases que nos permiten probar nuestras aplicaciones Java.El concepto fundamental en estas herramientas es el caso de prueba (test case), y la suite de prueba (test suite). Los casos de prueba son clases o módulos que disponen de métodos para probar los métodos de una clase o módulo concreta/o. Así, para cada clase que quisiéramos probar definiríamos su correspondiente clase de caso de prueba. Mediante las suites podemos organizar los casos de prueba, de forma que cada suite agrupa los casos de prueba de módulos que están funcionalmente relacionados. Las pruebas que se van construyendo se estructuran así en forma de árbol, de modo que las hojas son los casos de prueba, y podemos ejecutar cualquier subárbol (suite). De esta forma, construimos programas que sirven para probar nuestros módulos, y que podremos ejecutar de forma automática. A medida que la aplicación vaya avanzando, se dispondrá de un conjunto importante de casos de prueba, que servirá para hacer pruebas de regresión. Eso es importante, puesto que cuando cambiamos un módulo que ya ha sido probado, el cambio puede haber afectado a otros módulos, y sería necesario volver a ejecutar las pruebas para verificar que todo sigue funcionando.	
Ultima Versión	JUnit 4.4

7.6 Conclusión

Cuando decidimos realizar una aplicación Web, existen una gran cantidad de decisiones a tomar, decidir usar frameworks que den soporte a la arquitectura diseñada es una de ellas. En el momento de analizar los frameworks existentes nos encontramos con que la cantidad de opciones existentes en el mercado es abrumante. Si inicialmente, los frameworks nacieron con la idea de simplificar y mejorar ciertas tareas, su masiva proliferación complica mucho la tarea de selección, proporcionando, además un cierto grado de inseguridad al realizar la elección concreta para un determinado proyecto. Los siguientes enlaces pretender ser una muestra de la reflexión realizada:

http://java-source.net/open-source/web-frameworks
http://java-source.net/open-source/persistence

en estos enlaces se describe someramente un conjunto de los frameworks existentes. Otra muestra es el siguiente gráfico en el que se presenta un conjunto de aplicaciones y frameworks compatibles con la última especificación de Java:

La evolución que están sufriendo los frameworks Web indican una clara tendencia a la abstracción del protocolo en el que se sustenta (http) permitiendo beneficiarse de los modelos basados en componentes y controles que priman en las aplicaciones de escritorio. Las diferentes aproximaciones muestran que la mayoría de frameworks intentan resolver problemas entorno a los mismos conceptos: navegación, internacionalización, manejo de errores, validación de entradas, escalabilidad, portabilidad, acoplamiento etc. Los más antiguos, como Struts, admiten un manejo más directo de los datos que se procesan mientras que otros más modernos como JSF o Tapestry buscan la abstracción casi total del protocolo pero a cambio de generar modelos de componentes fácilmente extensibles y orientados a eventos.

Uno de los problemas sobre el que realizan un mayor hincapié es el problema de la separación de responsabilidades. Este punto se hace importante para tareas de desarrollo, mantenimiento, portabilidad, escalabilidad.... No todos lo solucionan de la mejor manera. Algunos frameworks no tienen en cuenta este tema mientras que para otros como Tapestry es uno de sus puntos fuertes o Cocoon que lo asume de forma muy estricta y basa su desarrollo es esta premisa.

Por otro lado comentar que aunque los frameworks buscan inicialmente solucionar problemas generales, encontramos un rendimiento mucho mas optimo cuando centran sus esfuerzos en solucionar problemas particulares. En este sentido pueden especializarse y ofrecer una solución más óptima.

El futuro próximo, influenciado por el nuevo modelo arquitectónico web 2.0, parece marcado por las interfaces ricas, una interactividad sin necesidad de realizar consultas constantes al servidor ni realizar recargas completas de página, admitiendo la posibilidad que desde el cliente se puedan acceder a diferentes servidores a la vez, lo que marca un cambio en la concepción del navegador web, que pasa de ser una interfaz sin inteligencia ni estado (salvo por las cookies) a otra con la posibilidad de procesar la información y mantener un estado propio.

Como reflexión final, expresar que el aspecto más interesante de conocer varios frameworks es saber que no existe uno que sirva de 'panacea' y que sea la solución a todos los problemas sino que cada uno fue desarrollado con objetivos diferentes y es necesario ver cual de todos se alinea mejor con los objetivos de nuestro proyecto evaluando ventajas y desventajas de cada uno.

8 COMPARACIÓN DE FRAMEWORKS

A continuación se realizará una explicación, más detallada, del funcionamiento de una par de frameworks para la capas de presentación, lógica de negocio y persistencia. De entre todos los frameworks estudiados, se han seleccionados los que, a partir de la información recopilada, parecen estar al frente de la tendencia actual para cada capa. Esta selección se ha hecho teniendo en cuenta temas relacionados con la madurez de la herramienta, su documentación, evolución tecnológica y los ejemplos de utilización teórica y práctica existentes.

En este punto no se pretende entrar en el detalle técnico profundo de cada framework, esto queda fuera del alcance del presente proyecto y además existe suficiente información en la Web. El objetivo es repasar los conceptos más importantes de cada uno de ellos.

Una vez estudiado cada uno de los frameworks se realizará una comparición entre ellos para encontrar los puntos fuertes y débiles entre ellos.

8.1 Capa de Presentación

Para la capa de presentación se han seleccionado los frameworks Struts y Java Server Faces.

8.1.1 Struts

8.1.1.1 Introducción

A principios de marzo del 2007 apareció en el mercado la primera versión disponible del nuevo framework Struts 2.0 (V 2.0.6 General Availability).Esta versión de Struts es la primera versión estable de la rama 2 de este framework. No se trata de una nueva versión de Struts 1.0, sino que se trata de una reescritura total del núcleo del framework para mejorar su antigua estructura, renovando y mejorando la anterior versión Para realizar esta nueva versión se han fusionado la primera versión de Struts con otro framework denominado WebWork de manera que se han aprovechado las ventajas de ambos frameworks y se ha rebautizado con el nombre de Struts 2. En el siguiente enlace se puede encontrar una comparativa de ambos frameworks.

Se trata de un frameworks de reciente aparición, con aún poca documentación y pocas pruebas de funcionamiento. Intenta acercarse a la filosofía de JSF pero considero que aún es demasiado pronto para evaluarlo. Además he encontrado algunas críticas como la publicada por Matt Raible en un artículo de su blog. La crítica se centra sobre OGNL (Object-Graph Navigation Language), un lenguaje de expresiones que emplea Struts 2.0 para navegar en el grafo de objetos y acceder a determinadas propiedades que queremos emplear en la generación de la vista. El problema es que si el desarrollador se equivoca y comete algún error al teclear el nombre completo de la propiedad (por ejemplo, teclea usurio.id en vez de usuario.id) el framework no genera ningún mensaje de error. De esta forma tareas de desarrollo y mantenimiento se plantean costosas. También se queja de que los usuarios del framework no proporcionan una realimentación adecuada a los desarrolladores por no quejarse de este problema y critica la falta de documentación. Además, no ha sido hasta la última versión V2.0.9, de reciente publicación, que no se han solucionado algunos bugs relacionados con el OGNL y con temas de seguridad.

Debido a lo anteriormente comentado he decido comentar la funcionalidad de Struts 1, ya que aunque se trata de un frameworks con bastantes años y, quizá un poco desfasado, ha sido ampliamente usado con un rendimiento bastante bueno.

8.1.1.2 Evolución

Craig R. McClanahan inició, en mayo de 2000, el proyecto Struts, con la idea de crear un marco de trabajo estándar, bajo el patrón MVC, para la comunidad Java. La historia de este framework es larga, y su presencia continuada durante muchos años en el mercado hace que posiblemente sea el framework más antiguo y con más uso. A continuación se resumen algunas de sus fechas más importantes

- Mayo de 2000. Struts 0.5. Craig R. McClanahan dona el código a Apache
- Junio de 2001. Struts 1.0. Versión inicial.
- Junio de 2003. Struts 1.1. Commons.
- Diciembre de 2004. Struts 1.2. Wildcard.
- 2006. Struts 1.3. Reorganización en subproyectos

Struts Component Framework está basado (se apoya) en las siguientes tecnologías:

- Java Runtime Environment (JRE) 1.4 or later.
- Servlet API 2.4 or later. (*)
- JavaServer Pages 2.0 or later. (*)
- JavaServer Faces 1.1 or later.
- JSP Standard Tag Library (JSTL) 1.1 or later.
- Apache Commons BeanUtils 1.7 or later.
- Apache Commons Chain 1.0 or later.
- Apache Commons Digester 1.7 or later.
- Apache Commons Logging 1.0.4 or later.

por tanto necesitaría, como mínimo, un servidor de aplicaciones J2EE 1.4. Aun así se podría hacer funcionar en un contenedor de Servlets 2.3 / JSP 1.2 (J2EE 1.3) usando la sintaxis apropiada en los descriptores de despliegue y páginas JSP y sustituyendo la librería JSTL 1.1 por una JSTL 1.0

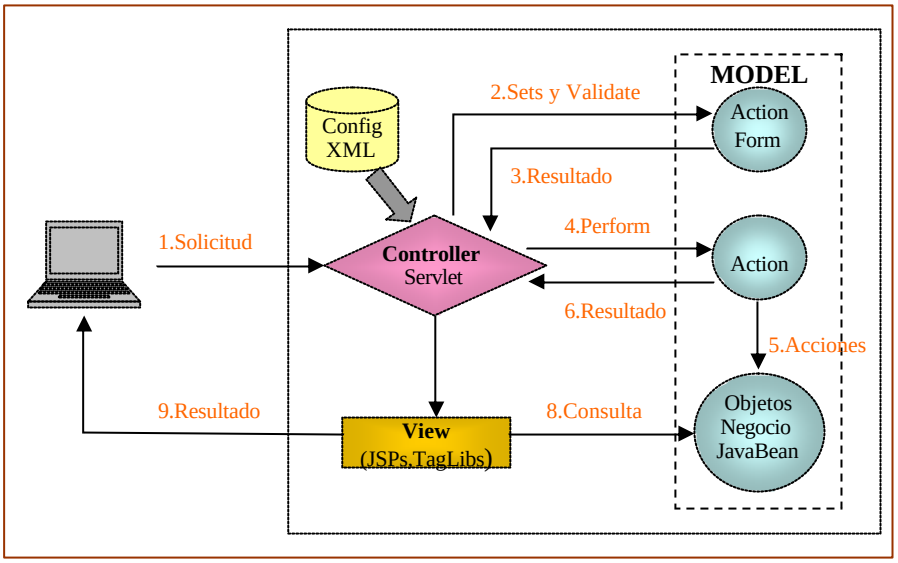
8.1.1.3 Visión Global

Para comprender el funcionamiento, es importante entender el flujo general de una petición desde que se genera en el navegador. Ésta se recibe en el lado servidor y es atendida por un objeto Controller (en realidad es únicamente una servlet especializada). Éste objeto contiene la lógica suficiente para analizar la solicitud, ver cual es la configuración indicada en el XML (indica la forma de actuar) y finalmente llamar al objeto Action correspondiente. Éste objeto es el encargado de instanciar y/o utilizar los objetos de negocio necesarios para concretar la tarea. Dependiendo del resultado retornado por Action, el Controller derivará la generación de interfaz a una o más JSPs, las cuales podrán consultar los objetos del Model (contenedores de los datos resultantes) a fines de realizar su tarea.

El controlador ya está implementado por Struts, aunque si es necesario se puede heredar y ampliar o modificar, y el flujo de trabajo de la aplicación se puede programar a través de un archivo XML Las acciones que se ejecutarán sobre el modelo de objetos de negocio se implementan basándose en clases predefinidas por el framework y siguiendo el patrón Facade. Y la generación de interfaz se soporta mediante un conjunto de Tags predefinidos por Struts con el objetivo de evitar el uso de Scriplets (los trozos de código Java entre "" dentro de las páginas), lo cual aporta ventajas de mantenibilidad y de rendimiento (pooling de Tags, caching, etc).

Su estructura consigue separar muy lo que se corresponde con la gestión del flujo de trabajo de la aplicación, del modelo de objetos de negocio y de la generación de interfaz. Además potencia la reutilización, soporte de múltiples interfaces de usuario (Html, sHtml, Wml, Desktop applications, etc.) y de múltiples idiomas, internalización, etc.

A continuación se presenta un diagrama que nuestra el flujo desde que se realiza una petición hasta que se ve el resultado:



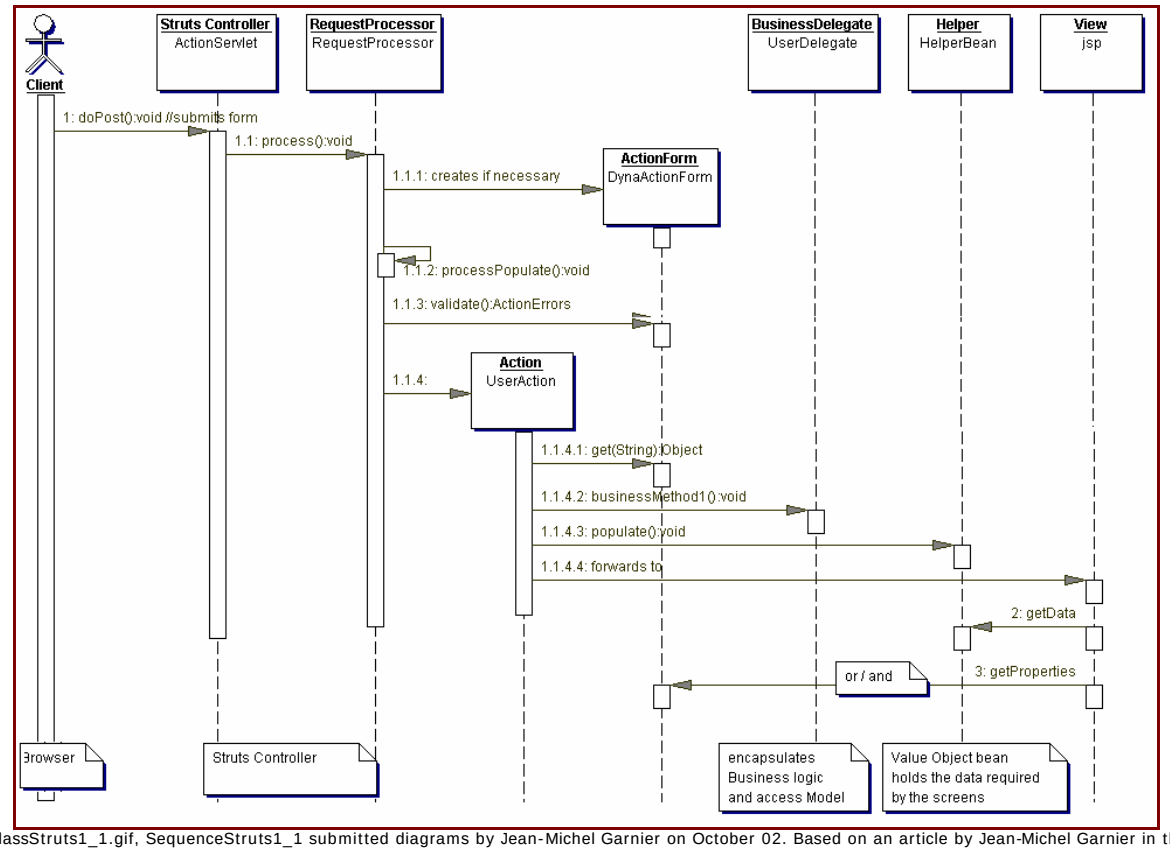
8.1.1.4 Lógica de navegación

Antes de entrar en detalle, es necesario comentar algunos conceptos importantes de este frameworks para facilitar la compresión posterior:

- ActionServlet, Es la parte del Controler que recibe la petición de cliente. delega el procesado de la petición en el objeto RequestProcessor (esta clase realiza acciones como el control de acceso, es decir, verificar que el usuario posee un rol de la lista de roles permitidos, lo realiza su método processRoles).
- Actions(UserActions), Objetos que heredan de la clase Action (propia de Struts) y que describen el proceso a realizar. Por ejemplo se puede decidir invocar alguna regla de negocio y en función del resultado mostrar la vista que corresponda.
- ActionForm, Encapsulan los parámetros de las peticiones de los clientes presentándolos como datos de un formulario. Representan los datos de entrada de la acción a realizar. Implementan las acciones asociadas con las comprobaciones y generación de mensajes de error. Un formulario puede compartirse entre varias peticiones de manera que se pueda ir llenando a partes antes de invocar la acción. Implementa el patrón Front Controller.
- ActionMappings, que permiten mapear las características de las acciones en un fichero externo XML. Asigna a cada Url en acciones (objeto asociado). Osea, se da un nombre a cada clase de acción para poder ser invocadas desde el cliente como un string.
- ActionErrors, para devolver mensajes de error en la validación de formularios ActionForms.
- Tiles, para creación y trabajo con plantillas (páginas Web).
- Struts Tags, librería de etiquetas que nos ahorrarán mucho código en la realización de páginas.
- Value Objects, objetos con los datos de soporte para la construcción de la páginas.
- UserDelegate, Se basa en el patrón Business Delegate. Se trata de una clase intermedia que desacopla los componentes de negocio del código que los usa, engloba los servicios empresariales y el acceso a la Modelo.
- HelperBean es un objeto que contiene los datos exigidos por la página. Patrón View Helper.

El Controller basa toda su inteligencia en las definiciones que se realizan en un archivo xml llamado struts-config.xml. En este archivo se guarda los mapeos de objetos (bean), todas las acciones asociadas y los formularios existentes con los que se trabajará.

A continuación se presenta el diagrama de flujo detallado para servir una petición cliente



Los pasos son los siguientes:

- 1. El cliente envía el formulario HTML
  - 1.1 El Controlador de Struts, Action Servlet, delega al objeto RequestProcessor el procesado de la petición.
  - 1.1.1Recupera y retorna el bean ActionForm asociado en el fichero de mapeo (action-mapping). Crea el bean si es necesario.
  - 1.1.2 Fusiona el bean ActionForm con los datos de los campos de entrada recibidos del cliente vía Html.
  - 1.1.3 Realiza la validación de los campos de entrada y crea los mensajes de error si el proceso de validación falla.
  - 1.1.4 Recupera una instancia del objeto UserAction para procesar la petición, llamando a su método execute (método sobrescrito de la clase base Action).
    - 1.1.4.1 Recupera datos del bean UserActionForm mediante los métodos codificados 'getProperties'
    - 1.1.4.2 Llama a Servicios de negocio a través del objeto BussinessDelegate.
    - 1.1.4.3 Opcionalmente puede utilizar otros objetos bean.
    - 1.1.4.4 Redirecciona a la página de destino especificada en el fichero xml de configuración (struts-config.xml).

La página retornada carga sus datos con :

- 2 El objeto HelperBean.
- 3 y/o del bean ActionForm.

8.1.1.5 Vinculación de datos (Binding)

El término "Binding" proviene del verbo inglés "bind" que significa atar, ligar o unir algo. En términos informáticos implica atar los datos. En el entorno de aplicaciones Web significa que existe una atadura entre las páginas Html y los objetos Java.

Paralelamente a la parte servidora, donde tenemos una servlet que actúa de controlador y se encarga de todas las tareas, nos encontramos con que el lado de las vista (cliente), struts ofrece una serie de bibliotecas de etiquetas (TAGs) que se insertan dentro del código html, facilitando el acceso a los beans en el momento de generación de las vistas. Podemos agrupar estas etiquetas en cuatro grupos:

- struts-bean: manejo de los beans a los que tiene acceso la página.
- struts-html: renderiza los componentes html comunes.
- struts-logic: permite direccionar el flujo de ejecución de la página según condiciones.
- struts-nested: permite el anidado de componentes.

Incluyendo estas etiquetas en el diseño de una página, es posible generar vistas dinámicas, sin necesidad de añadir código java(<%..%>). Para relacionar los datos de los objetos java y las vista se deben seguir las siguientes normas:

- En la vista, cuando una etiqueta necesite contener una valor accesible desde el action deberá tener un nombre igual al del form asociado con el action. De esta forma, Struts realiza el la asociación automáticamente entre el valor del tag y el valor del atributo del formulario. Por ejemplo, si en el formulario html queremos que una entrada de texto este asociado automáticamente con su bean deberíamos:

`<html:text property="dato">`	En el formulario
`public string getDato() public string setDato()`	En el bean de formulario asociado al action

- Si desde la vista se desea acceder a valores (objetos, propiedades, etc) establecidos desde el action. Se puede utilizar las etiquetas correspondientes para acceder a los objetos. Es posible navegar a través de los objetos mediante un lenguaje especial para acceder a sus propiedades. Por ejemplo, si se desea recorrer una colección de tiendas y el nombre de su responsable imprimiendo sus valores se podría hacer así

```
<table>
<logic:iterate id="unatienda" name="tiendas" scope="request" type="com.empresa.Tienda" >
    <tr>
      <td><bean:write name="unatienda" property="dato" /></td>
      <td><bean:write name="unatienda" property="responsable.dato" /></td>
    </tr>
 </logic:iterate>
</table>
```

  En este caso, en el bean del formulario debe existir un método *getTiendas* que posea una colección de objetos de la clase Tienda. La Tienda a su vez deberá tener un método *getResponsable* que devuelva un objeto que tenga un método llamado *getDato*. A cada ítem de la colección se le hace referencia dentro del tag *bean:write* a través del nombre una Tienda.

8.1.1.6 Internacionalización

Struts da soporte a la creación de aplicaciones multilenguaje. Para crear una aplicación de este tipo, debemos seguir los siguientes pasos.

- Crear un archivo de texto con la extensión 'properties' (NombreArchivo.properties) en el directorio donde se encuentren las clases de la aplicación. Este fichero debe contener los pares clave – valor con el formato 'clave.subclave=valor' (la primera parte será la que se codifica en la página y la segunda

es lo que realmente se muestra al usuario). El contenido de este fichero se corresponde con el idioma principal de la aplicación.

- Para cada nuevo idioma que queramos tener basta con crear un nuevo archivo (como antes) con el nombre 'NombreArchivo_xx.properties', donde xx corresponde al código ISO del idioma.
- Para configurar el idioma principal de la aplicación se deben configurar las entradas adecuadas del fichero struts-config.xml. La etiqueta 'servlet/init-param' poniendo como 'param-name' application y como 'para-value' la localización completa del archivo NombreArchivo.properties (sin extensión).
- Para usar esta técnica en las etiquetas basta con utilizar <bean:message key="clave.subclave"/>, donde clave y subclave se sustituirá por su valor asociado en el fichero '.properties' en función del idioma configurado para el usuario.

8.1.1.7 Validación de Entradas

Struts ofrece la posibilidad de validar los campos de entrada que ha introducido el usuario. Se ofrecen dos maneras de realizar este proceso:

- Sobrescribir el método 'validate()' del ActionForm asociado. Éste método es llamado por el servlet controlador después de que hayan rellenado las propiedades del bean, pero antes de que el sistema invoque al método 'perform()'.En caso de encontrar algún error de validación, se leen las claves de error a mostrar (ActionError) y se retorna el formulario al cliente con los mensajes indicados.
- Por otro lado, se pueden agregar validaciones a los campos de los formularios. Esta validaciones se ejecutan tanto n el lado cliente (utilizando javascript) como en el lado servidor. Además se permite definir las rutinas de validación más usadas. Para realizar la configuración es necesario agregar las reglas de validación en el archivo de configuración 'validation-rules.xml' y expresando las restricciones de los campos de cada formulario en el archivo 'validation.xml'.

Es importante que el último sistema, permite crear, modificar y eliminar las validaciones sin necesidad de tocar para nada el código java. Esto facilita mucho las tareas de mantenimiento.

8.1.1.8 Maquetización

La maquetación de la aplicación WEB es facilitada a través de Struts Tiles, un plugin que permite componer a partir de porciones de página, la página definitiva que será enviada al cliente. La composición de las partes se puede definir de tres maneras:

- A través de un xml
- Dentro de las páginas jsp
- Programáticamente desde las Actions

Algunos aspectos interesantes para mencionar son el soporte de internacionalización (composición de partes según el Locale); las composiciones se pueden heredar y redefinir; es posible tener varias composiciones y seleccionar una de acuerdo a una clave.

8.1.1.9 Independencia del motor de visualización

Struts en principio es independiente del motor de visualización aunque generalmente se elija jsp para mostrar las vistas. Existen formas de que struts envíe las vistas para que sean procesadas por motores de plantillas como velocity, transformadores de estilos de documentos XSLT u otros frameworks de presentación como JSF. Por lo tanto struts no está ligado a un motor de visualización particular sino que puede convivir con varios de estos, incluso utilizándolos en simultaneo.

8.1.2 Java Server Faces

La explicación de algunos de los conceptos interesantes de este framework se realizará desde un punto menos técnico al anterior. Esto es porque esta compuesto por una estructura más extensa y sofisticada. Realizar una explicación detallada llevaría muchas hojas y no es el objetivo del presente estudio. Además existe mucha información disponible en la Web.

8.1.2.1 Introducción

Es el estándar desarrollado por Sun para la capa de presentación Web. Forma parte de la especificación J2EE 5, que deberán cumplir todos los servidores de aplicaciones, y se define como una evolución natural de los frameworks actuales hacia un sistema de componentes.Es sencillo y aporta los componentes básicos de las páginas web además de permitir crear componentes más complejos (menús, pestañas, árboles, etc). Ya hay disponibles diferentes implementaciones de la especificación, tanto comerciales como de código abierto (MyFaces...), así como librerías de componentes adicionales que amplían la funcionalidad de esos componentes iniciales (Facelets).

Este framework ha sido muy bien acogido por la comunidad. Muchos de los proyectos de código abierto y las compañías con más influencia lo han identificado como el framework de presentación web del futuro. JBoss ha integrado directamente JSF con EJB3 mediante el proyecto Seam (abanderado además por Gavin King, el líder del equipo de desarrollo Hibernate). IBM lo ha incorporado como mecanismo de presentación estándar para su entorno, desarrollando no sólo el soporte completo en su IDE, sino nuevos componentes. Oracle es otra de las compañías que más ha apostado por esta tecnología, ofreciendo la que posiblemente sea la mayor oferta de componentes propios.

Dentro de los 'pesos pesados' en Java, nombres que han guiado a la industria en el pasado como Matt Raible, Rick Hightower, David Geary, el mencionado Gavin King y por supuesto el propio creador de Struts y ahora arquitecto de JSF, Craig McClanahan.

8.1.2.2 Evolución

Inicialmente liberado en Marzo de 2004, JavaServer Faces 1.0 fue rápidamente seguido por una versión 1.1 de mantenimiento que eliminaba algunos errores tipográficos e inconsistencias iniciales, y también corregía algunos graves errores en la correspondiente implementación de referencia.

JavaServer Faces 1.2 está en desarrollo (bajo la atención del grupo de expertos de las especificaciones JSR-252). La mayor parte del esfuerzo de desarrollo se está enfocando en mejorar el alineamiento entre JavaServer Faces y JavaServer Pages (versión 2.1 que también está desarrollándose), particularmente en las áreas de resolución de diferencias entre la sintaxis y la semántica del lenguaje de expresión, y los problemas de interoperabilidad entre los componentes JavaServer Faces y las plantillas de texto JSP.

Cuando esté terminado, el soporte de JavaServer Faces 1.2 será obligatorio en cualquier plataforma Java 2 Enterprise Edition (J2EE) 5.0: las aplicaciones basadas en J2EE 5.0 podrán asumir que el servidor de aplicaciones soportará JSF. Mientras tanto, JavaServer Faces ya ha ganado el apoyo de los desarrolladores (incluyendo aquellos que están creando librerías de componentes personalizados sobre los APIs estándar), así como robustas herramientas de soporte como Oracle JDeveloper 10g, Sun Java Studio Creator, e IBM WebSphere Application Developer, Eclipse. Está claro que esta tecnología será incluso más mayoritaria en un futuro.

8.1.2.3 Visión Global

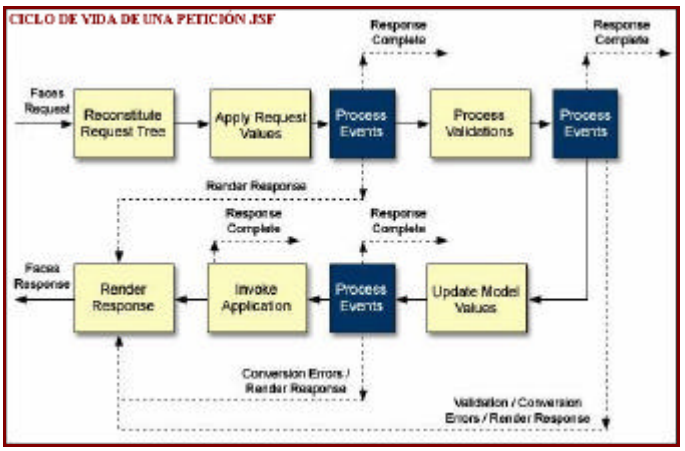
JSF también basa su funcionamiento es un controlador central (FrontControler) cuya función consiste en procesar todas las peticiones del cliente y gestionar su ciclo de vida. Está basado en un modelo de componentes para la interfaz de usuario. Un componente JSF es un elemento rehusable y configurable que

se puede utilizar en la interfaz de usuario. Los componentes se pueden anidar. Por ejemplo, una página contiene un componente mapa que a su vez contiene un componente botón. El diseño del framework permite navegar a través del árbol de componentes para acceder a sus propiedades. Además, los componentes pueden reaccionar a diferentes eventos y son capaces de almacenar su estado interno. Además, JSF permite definir la lógica de navegación a través de reglas de navegación especificadas en uno o más archivos de configuración (faces-config.xml), en este archivo se definen cosas como que viniendo desde una página, se realice una determinada acción de un objeto y dependiendo del resultado se redirija el flujo a la página indicada.

Un aspecto fundamental de JSF es el ciclo de vida de una petición web. El ciclo de vida está separado en 6 fases principales.

8.1.2.4 Lógica de navegación

El ciclo de vida de una petición **JSF**, son los diferentes caminos por donde irá pasando la petición realizada. Dependiendo de ciertos factores, este ciclo puede ser interrumpido o modificado. A continuación se presenta el ciclo de una petición estándar:



- ***Reconstruir el árbol de componentes***: Es esta primera fase se empieza a construir el árbol de componentes, llama al objeto UIViewRoot contenido en el FacesContext (objeto que contiene toda la información de estado que JSF necesita para manejar el estado de los componentes visuales para la petición actual en la sesión actual), del cual se obtendrá la información referente a la página. Además se conectan los manejadores de eventos, los validadores y se graba el estado.
- ***Aplicar valores de petición***: una vez construido el árbol, se asocian los valores que se encuentran en la petición, con el método 'decode' a los componentes. Se comprueba el valor de la propiedad 'inmediate' que tienen los componentes. Esta propiedad sirve para capturar eventos y validaciones en esta fase del ciclo. El valor de esta propiedad es un Booleano*:*
    - *Falso:* Los valores sólo se convierten, es decir, si para un campo se esperaba un entero se convierte el valor recogido a entero. Si se produce un error se guarda en el 'faces Context' y lo mostrará durante la última fase.
    - *Cierto:* Los valores se convierten y se validan. Si hay eventos asociados se capturan también. Igual que anteriormente si se produce un error se mostrará en última fase. La consecución de un evento puede forzar el paso a la última fase, de esta forma se valida / convierte sólo una pequeña parte de la página.
- ***Procesar Validaciones***: Durante esta fase, la implementación JavaServer Faces procesa todas las validaciones registradas con los componentes del árbol. Examina los atributos del componente que especifican las reglas de validación y compara esas reglas con el valor local almacenado en el componente. Si se produce algún fallo en la validación se corta el ciclo y se pasa directamente a la última fase. Los mensajes de error generados se guarda en el FacesContext.

- **Actualizar los valores del Modelo**: una vez que se determina que el dato es válido, se actualizan el valor de las propiedades de los componentes en la clase Java.
- **Invocar aplicación**: durante ésta fase, la implementación JavaServer Faces maneja cualquier evento a nivel de aplicación, como enviar un formulario o enlazar a otra página.
- **Renderizar la respuesta**: durante ésta fase, la implementación JavaServer Faces invoca las propiedades de codificación de los componentes y dibuja los componentes del árbol de componentes grabado en el FacesContext.

8.1.2.5 Vinculación de datos(Binding)

El modelo de componentes permite el enlace de valores (value binding), de métodos (method binding) y de componentes (component binding).

- *Binding de* Valores:
  Cualquier componente de interfaz admite enlazar sus atributos y propiedades con valores de alguna propiedad (o expresión) de algún bean. Para enlazar los valores con los atributos se debe encerrar entre #{} el campo que se desea enlazar. La página mostrará al usuario el valor contenido en la propiedad asociada.

  Para poder enlazar correctamente los valores de un componente de interfaz con los de un bean, las propiedades que se enlazan tienen que ser de tipos compatibles o debe haber un convertidor (Converter) asociado. JSF provee un amplio set de convertidores pero también es posible definir nuevos.

- *Binding de Métodos*
  También es posible enlazar métodos, permitiendo su ejecución con el envío de parámetros, si es necesario, y con la recepción, si existe, de una respuesta. La cantidad de parámetros y la respuesta están determinadas por el tipo de método que se espera.

- *Binding de Componentes*
  Por último, el enlace de componentes sirve para vincular directamente un componente de interfaz con una propiedad de un bean de manera de poder manejarlo programáticamente. El vínculo se realiza a través del atributo binding de los componentes que lo tienen. El objeto enlazado ser una propiedad de un bean que pueda leerse y escribirse y debe descender de UIComponent.

8.1.2.6 Gestión de Eventos

Los componentes ofrecidos por JSF están orientados a eventos y por tanto disponemos de la funcionalidad para manejar eventos que se producen en el lado cliente (cambiar el valor de un textbox o hacer clic en un botón). Todos los eventos son gestionados en el lado servidor.

Todos los eventos generados desde los componentes de la interfaz son subclases de FacesEvent. Las dos subclases estándares que derivan de FacesEvent son ActionEvent y ValueChangeEvent. Action event generalmente se utiliza cuando se refleja el uso de un control, como por ejemplo, el presionar un botón. ValueChangeEvent se utiliza cuando se quiere reflejar el cambio de algún valor de importancia en el componente. En el lado servidor disponemos del interfaz FacesListener que define los métodos básicos para poder escuchar eventos. ActionListener y ValueChangeListener que son las implementaciones correspondientes para los eventos comentados anteriormente.

8.1.2.7 Internacionalización

Esta funcionalidad permite mostrar la aplicación según el idioma del usuario. Para ello en el fichero faces-config.xml vamos a especificar que localizaciones soporta la aplicación, y cual es la localización por defecto. La internacionalización de JSF está construida sobre la base del soporte que ya brindaba java, la especificación de Servlets y y la de JSPs.

8.1.2.8 Validación de Entradas

La lógica del proceso de validación de la información de usuario reside en los Validadores (Validators). Un validador se encarga de realizar comprobaciones sobre el valor un componente durante la fase Process Validations. A Cada componente que recibe la entrada de valores por parte del usuario se le pueden registrar 0 o más validadores. También es posible llamar a los validadores en cualquier momento a través del validate() del componente.

JSF incluye varios validadores estándar pero también permite crear validadores nuevos implementando una interfaz y definiendo los atributos que se utilizarán para configurarlo. Además de las validaciones comentadas, es posible realizar validaciones a nivel de aplicación o negocio. Indicar reglas asociadas a la propia lógica de negocio que deben cumplirse y que se ejecutan a través de métodos.

8.1.2.9 Construcción de Componentes

JSF admite la creación de componentes propios con o sin render asociado. Sin embargo, la creación no es sencilla y se necesita crear varios archivos dependiendo el tipo de componente que se desea crear. No se entrará en detalles sobre como crear componentes con JSF, existe mucha información consultable en la web sobre estos temas más técnicos.

8.1.2.10 Independencia del motor de visualización

La codificación de los valores de los componentes para ser mostrados en la vista y la decodificación necesaria de los valores que llegan de las peticiones varía dependiendo del dispositivo. En JSF, esta codificación / decodificación se puede realizar de dos maneras, utilizando un modelo de implementación directa o utilizando un modelo de implementación delegada. En el modelo de implementación directa cada componente posee la lógica para codificarse y decodificarse a si mismo. En cambio, cuando se utiliza el modelo de implementación delegada, esta lógica se deposita en el "Renderizador" que cada componente tiene asociado y que se especifica en la propiedad RenderedType. Con la primera opción se facilita la creación de componentes pero con la segunda se pueden crear componentes que, dependiendo la situación, se le presenten al usuario de diferente manera. Por ejemplo se puede indicar que se utilice un Renderizador determinado para las peticiones que provienen desde un móvil o que se presenten de manera especial para un idioma predeterminado.

8.1.3 Comparativa

A continuación se describirán una serie de aspectos relevantes y diferenciadores entre los dos frameworks.

Struts

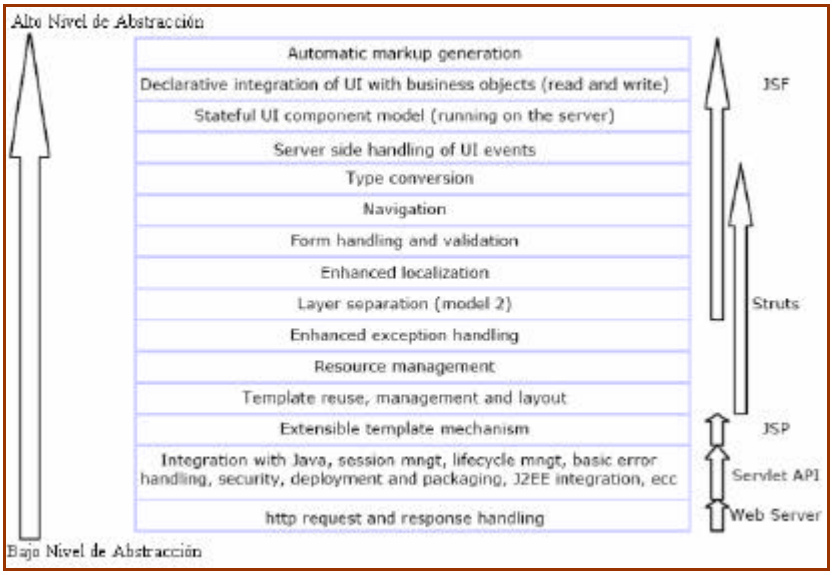
- Puntos Fuertes
  - Muchos años de madurez en el mercado.
  - Mucha documentación y aplicaciones realizadas exitosamente.
  - Curva de aprendizaje mediana.
  - Open Source (Licencia Apache).
- Puntos Débiles
  - Diseño anticuado.
  - No es una especificación
  - Codificación tediosa (ActionForms deben extender clases Struts).
  - No tiene una óptima separación de capas. Presenta cierto grado de acoplamiento.
  - No admite la creación de componentes propios.
  - No admite renderizar automáticamente vistas en función del dispositivo de visualización.

Java Server Faces

- Puntos Fuertes
    - Sólo tres años de madurez pero con una comunidad de desarrollo creciente muy importante. Empresas como IBM o arquitectos como Craig McClanahan son un claro ejemplo.
    - Es una especificación por lo que admite implementaciones de terceros. Esto permite no atarnos con un proveedor y poder elegirlo según: componentes ofrecidos, rendimiento, soporte, precio, política.
    - Es mucho más flexible ya que al estar orientado a componentes y dar soporte a eventos ofrece un abanico de posibilidades mayor.
    - Tanto Struts como JSF son muy flexibles en las reglas de navegación, pero JSF permite más flexibilidad y un mejor diseño porque las reglas de navegación están desacopladas de las acciones.
    - Permite crear componentes propios.
    - Admite renderización automática en función del dispositivo de visualización.
    - Separa completamente los contenidos de la capa de presentación de la lógica.
    - Permite modificar el comportamiento de la aplicación sin tocar el lenguaje de implementación.
    - Obligatoriamente soportado en cualquier plataforma J2EE 5.

- Puntos Débiles
    - La creación de componentes propios es compleja.
    - Requiere javascript.

El siguiente cuadro muestra una evolución comparativa:



8.1.4 Conclusión

Aunque Struts es el framework que ofreció un camino hacia el MVC en Java para desarrollos Web, es también uno de los que más ha acusado el paso de los años en tiempo de desarrollo y flexibilidad. Hoy existen frameworks que comparten los mismos principios pero con más potencia, elegancia y flexibilidad. Este es el caso de JSF, aunque tiene una curva de aprendizaje más elevada que Struts, pero todo lo que aporta justifica sobradamente este esfuerzo. Una vez hemos superado la fase de aprendizaje, el diseño de JSF facilita mucho las tareas de gestión y mantenimiento de nuestra aplicación. Además, al tener un alto desacoplamiento entre la capa de presentación y la lógica de negoció, ofrece la posibilidad de realizar cambios de forma más fácil, eficaz y rápida.

8.2 Capa de Lógica de negocio

La evolución actual de las herramientas que dan soporte a la lógica de negoció tiende hacia una filosofía de simplificación. Cada vez se intentan construir modelos arquitectónicos más sencillos, sistemas demasiados amplios se vuelven complejos y en la mayoría de casos aportan más carga de trabajo que beneficios. Los arquitectos deben seleccionar que tecnología es la más apropiada para un contexto de negocio determinado, basado en criterios como: la estandarización, la especificación, etc.

Actualmente dos de las principales tecnologías para el desarrollo de soluciones empresariales son el framework Spring y EJB 3.0 (no así versiones en sus anteriores versiones).

8.2.1 EJB 3.0

8.2.1.1 Introducción

Hace ya una década que se desarrolló desarrollo el concepto de EJB. Nació como un marco de trabajo para el desarrollo de aplicaciones empresariales en Java, dando soporte a la implementación de la capa modelo de una aplicación. Durante estos años ha ido sufriendo muchos cambios y modificaciones.

Los EJBs proporcionan un modelo estándar de componentes distribuidos en el lado del servidor. Su objetivo es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad,...) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes nos permite que éstos sean flexibles y sobre todo reutilizables. No hay que confundir los Enterprise JavaBeans con los JavaBeans. Los JavaBeans también son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones, pero no pueden utilizarse en entornos de objetos distribuidos al no soportar nativamente la invocación remota (RMI).

En sus últimas versiones, el modelo de EJB está definido a través de un JSR (Java Specification Requerimient). Se basa en una especificación, o sea, un modelo de programación, que debe ser implementada por cualquier proveedor de servidor de aplicaciones que desee ser compatible con la misma. La especificación de EJB define una arquitectura para el desarrollo y despliegue de aplicaciones basadas en objetos distribuidos transaccionales, software de componentes en el lado del servidor. Las organizaciones pueden construir sus propios componentes o comprarlos a terceros. Estos componentes existentes en el lado servidor, llamados Beans Enterprise, son objetos distribuidos que están localizados en contenedores de JavaBean Enterprise y proporcionan servicios remotos para clientes distribuidos a lo largo de la red.

La evolución de los EJB sufrió un momento crítico con la aparición del modelo de programación propuesto por la versión 2.1. Ésta comportaba una serie de inconvenientes (dificultad de uso, falta de potencia en el soporte de persistencia...) que limitaron mucho el uso de esta especificación y motivó la aparición de soluciones open source que suplían las carencias que presentaba EJB 2.1. A día de hoy, parece que las soluciones de este tipo que más han marcado el desarrollo empresarial dentro de la plataforma Java han sido Hibernate y Spring Framework. La nueva versión 3.0 de EJB pretende dar solución a muchos de los problemas que presentaba, para ello ha incorporado muchas de las características de estos frameworks para procurar a los desarrolladores una plataforma de desarrollo bastante más sencilla que su predecesora versión

8.2.1.2 Evolución

En este framework, éste es un punto importante ya que, al iniciarse su desarrollo hace muchos años, convine ver su evolución a lo largo del tiempo. La especificación EJB ha ido evolucionando a la par que lo hacía la propia especificación J2EE. Las diferentes versiones que han existido hasta la fecha son:

- Marzo de 1997 – Se dispone de la primera especificación de EJB y es desarrollada por IBM.
- Marzo de 1998 - EJB 1.0: la especificación original. Adoptada por Sun Microsystems.
  - Define los roles que cada componente EJB debe asumir en el modelo arquitectónico
  - Define la vista cliente de los beans de negocio.
  - Define las responsabilidades del contenedor EJB y del servidor que lo soporta. Define el sistema necesario para dar soporte a su desarrollo y ejecución.

- Diciembre de 1999 - EJB 1.1: la primera incluida dentro de J2EE.
  - Cambia los descriptores de desarrollo XML.
  - Admite contexto JNDI. Admite RMI sobre IIOP.
  - Seguridad a nivel de rol.
  - Soporte obligado a los beans de entidad (Entity Beans).
  - Ofrece un mejor soporte al desarrollo y ensamblado de aplicaciones.
  - Ofrece una especificación más completa de los diferentes roles de cada EJB.

- Agosto de 2001 - EJB 2.0: incluida en J2EE 1.3, añadía las interfaces Locales y los Message-Driven beans. A partir de esta versión inicia el cumplimiento de las especificaciones de la comunidad Java Community Process. Esta versión sigue la especificación JSR 19.
  - Admite el desarrollo de una aplicación distribuida combinando componentes desarrollados por herramientas de diferentes vendedores.
  - Facilita el desarrollo de aplicaciones. Abstrae al desarrollador de conceptos de bajo nivel como el control de transacciones, pool de conexiones, multi-threading y otras Apis complejas.
  - Sigue la filosofía de Java, 'Write Once, Run Anywhere'. O sea, se desarrolla una vez y se ejecuta en diferentes plataformas sin necesidad de modificar el código ni recompilar.
  - Concreta aspectos del ciclo de vida de los objetos.
  - Se hace compatible con plataformas de servidor existentes. Producto se extienden para ser compatible y soportar EJB.
  - Se compatibiliza con otras Apis Java.
  - Compatible con protocolos CORBA.

- Noviembre 2003 - EJB 2.1: Se incluye en J2EE (revisión 1.4). Sigue la especificación JCP JSR 153.
  - Soporte a Web Services: los beans de sesión pueden ser invocados por el protocolo SOAP/HTTP. Además también puede invocar fácilmente Web Services.
  - Mecanismo, basado en eventos, que permite invocar EJB cada cierto tiempo (Timer Service).
  - Los beans de mensajes se mejoran para aceptar otras entradas además de JMS.
  - Nueva funciones del lenguaje de consultas (EJB-SQL): order by,avg,min,max,sum, count, mod.
  - Se pasa a utilizar esquemas XML para especificar los descriptores de despliegue (no DTD).

- Mayo 2006 - EJB 3.0: Ahora con anotaciones, Cluster (soporte a redes de ordenadores) y esta incluida en J2EE 5.0. Sigue la especificación JCP JSR 220. Su objetivo es simplificar la implementación de beans (se eliminan interfaces y ficheros xml) sin por ello sacrificar su potencia y flexibilidad además de facilitar la implementación de persistencia mediante el uso de anotaciones.

8.2.1.3 *Conceptos Básicos*

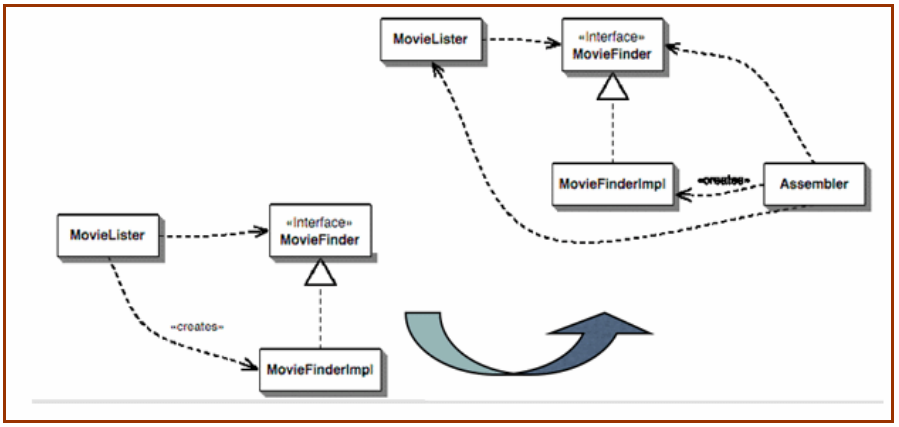
EJB es el único estándar Java que se ocupa de la lógica de negocio del lado del servidor, y esto es fundamental para J2EE. El comité de especificación de EJB reconoce que EJB se quedó corto en alcanzar algunas de sus ambiciosas metas, y necesitó de una modernización. Esta modernización está muy enfocada en la facilidad de uso, principalmente mediante la simplificación de los requerimientos para los desarrolladores. Además, el comité de especificación también ha identificado un número de mejoras funcionales críticas que facilitan el uso y la eliminación de ciertos anti-patrones J2EE.

- ***Simplicidad de Uso***: Elimina la necesidad para los interfaces Home y para los beans de implementar los requisitos de las Apis EJB (no implementen la interfaz javax.ejb.EnterpriseBean). Por tanto , los Bean Session, Message driven Beans, y Entity Beans son ahora objetos simples (POJOs) focalizando mucho más su atención en la lógica de negocio. Los beans son ahora clases java estándar, evitando los anteriores requerimientos de objetos pesados, por tanto pueden seguir principios y beneficios de la orientación a objetos como la herencia, el polimorfismo y otros estándares de construcción
- ***Java Annotations:*** Se simplifica mucho la configuración a través de valores por defecto y del uso de anotaciones de metadatos (datos que describen otros datos) en lugar de los anteriores descriptores de despliegue en XML. Éste concepto está definido dentro de su especificación y está soportado por JDK 5. Una anotación es una forma de añadir metadatos al código fuente Java que quedan disponibles en tiempo de ejecución del programa. Pueden añadirse a elementos de programa tales como clases, métodos, campos, parámetros, variables locales, y paquete.

  La especificación EJB 3 ofrece un gran abanico de posibilidades en el ámbito de las anotaciones.También se admite no usar está técnica y utilizar descriptores de despliegue XML. Servidores de aplicación como JBoss y herramientas de persistencia como Hibernate soportan el concepto de anotación, lo que significa que se pueden tener todos los beneficios de EJB fuera del contenedor de objetos EJB3 y dentro de aplicaciones Java independientes.

- ***Dependency Injection:*** Éste termino hace referencia a un concepto de diseño relativamente nuevo. Se aplica cuando queremos tener un conjunto de objetos que 'dependen' unos de otros pero que a la vez, para estar desacoplados, no se conocen (ninguno referencia a los otros, aunque bien podrían hacerlo), un sistema externo sabe como debe de vincularlos.

  Para comprender este patrón, supongamos que tenemos un diagrama de clases en el que la Objeto A necesita una instancia de un Objeto B para realizar algún tipo de proceso. Tradicionalmente esto se realiza utilizando dentro del Objeto A un método constructor del Objeto B, pudiendo acceder así a sus propiedades y métodos. Esto no es necesario usando Inyección de dependencia, lo único que hay que hacer es definir esta relación de objetos y, en tiempo de ejecución, un proceso externo (ensamblador), proporcionará automáticamente al Objeto A un Objeto B con los datos adecuados. La siguiente figura muestra la diferencia entre el método tradicional de instanciar objetos y la técnica de inyección de dependencia.



- ***Tipos de Bean:*** En este punto se explicará superficialmente cuales son los distintos tipos de Enterprise Java Bean y cuáles son los propósitos de cada uno de ellos: Stateless Session Bean, Stateful Session Bean, Entity Bean y Message Driven Bean. Debido a que éste pretende ser una breve explicación, no se entrará en detalles tales como ciclo de vida de los mismos y otros tipos de características.

    - *Stateless Session Bean*:Un bean de sesión sin estado es aquél que no dispone de variables de instancia en las cuales se guarden datos que puedan ser compartidos entre los distintos métodos del bean. Es decir, se trata de un bean que por lo general contará con una serie de métodos que realizarán un trabajo determinado e independiente y que el resultado de las operaciones realizadas dentro de cada uno de los métodos no dependerá de ningún estado relativo a la conversación que mantiene el cliente con el bean.
    - *Stateful Session Bean*: Al contrario que en los beans de sesión sin estado, estos sí tienen estado. Esto significa que dentro de la sesión del usuario estos beans van a almacenar datos en variables de instancia, y esos datos van a tener un significado concreto durante toda la conversación mantenida entre el cliente y el bean. Un ejemplo típico es un carro de la compra, en el cual, durante la sesión de usuario, éste va agregando productos en el carro a través de una lista. Tras ir agregando productos llegará un momento en el que el usuario quiera efectuar la compra, para lo cuál tendrá que realizar previamente un registro de sus datos, especificar la dirección de entrega, indicar el número de su tarjeta de crédito, etcétera, y finalmente confirmará la compra de los productos seleccionados. Como vemos, todos estos datos hay que almacenarlos en unas variables, que son las que conforman el estado del bean.
    - *Entity Bean*: Un Entity Bean es una clase (POJO) que representa una tabla de una base de datos, y cada instancia de esta clase representa un registro de la tabla, es decir, con los entity beans lo que conseguimos es crear un mapeo entre las propiedades de una clase y los campos de una tabla. Además, también vamos a poder especificar las relaciones que tienen las clases entre sí (uno a uno, uno a muchos, muchos a uno y muchos a muchos). Todo Entity Bean debe de tener una clave primaria que identifica a ese registro de forma única dentro de la tabla. Todas estas configuraciones las vamos a realizar a través de anotaciones, y el API que se encarga de gestionar todos los aspectos relativos a la persistencia es JPA (Java Persistent API).
    - *Message Driven Bean*: Los beans (MDB) permiten a las aplicaciones procesar mensajes de forma asíncrona a través del servicio JMS (Java Messaging Service). Este servicio funciona a través de colas de mensajes, que es donde los clientes envían sus peticiones, y estas colas son controladas por los Message Driven Beans, los cuales procesan los mensajes que hay en ellas y ejecutan ciertos servicios dependiendo del mensaje procesado. Son beans sin estado.

- ***Entity Manager API:*** Este Api se introduce en EJB 3.0 y está diseñada para permitir a las instancias de objetos Java Bean ser seleccionados y actualizados de forma local, y posteriormente se enviaran al EntityManager para ser persistidos en la base de datos. Permite crear, encontrar por clave primaria y eliminar objetos persistentes. Además permite crear objetos 'Query' para realizar consultas.
- ***Simplificación de Persistencia y Mejora de Consultas de datos:*** Estandariza el modelo de persistencia de Java con Hibernate jugando un papel importante en la definición de este Api. Se ha provisto un gran conjunto de anotaciones para tratar el mapeo de objetos relacionales y todo el rango de diferentes tipos de relación entre ellos. Además ha mejorado enormemente EJB-QL soportando consultas dinámicas, consultas anidadas, borrados y actualizaciones en masa ..., y añade soporte para la ejecución de SQL nativo.
- ***Compatibilidad con EJB 2.x:*** EJB 3.0 requiere compatibilidad e interoperabilidad con EJB 2.x; las aplicaciones EJB 2.x funcionan en contenedores compatibles con EJB 3.0. El modelo de programación para los EJB ha cambiado drásticamente entre EJB 2.x y EJB 3.0, por lo que migrar una aplicación a la nueva especificación no resulta trivial.

Los vendedores de herramientas y de servidores de aplicaciones proporcionarán herramientas y utilidades para hacer la migración más fácil.

8.2.2 Spring

8.2.2.1 Introducción

Spring es un framework de aplicaciones Java/J2EE desarrollado usando licencia de OpenSource. Permite aumentar la productividad, ayudando a realizar las tareas repetitivas pero con diseños limpios y consistentes. Su enfoque es maximizar el uso de patrones IoC junto con un diseño basado en interfaces en lugar de clases.

Su efecto y su influencia en los desarrollos posteriores ha sido notable. Spring popularizó los términos IoC, DI desplazando al MVC de las lista de patrones más destacados. Aunque no se use Spring en un determinado desarrollo, es probablemente que su influencia se aprecie en el desarrollo de cualquier proyecto que se empiece en los próximos años.

Entre otras cosas Spring puede organizar de forma efectiva nuestros objetos eliminando la proliferación de Singletons y facilitando las buenas prácticas de programación orientada a objetos (utilizando interfaces), utiliza AOP (aspect-oriented programming) para ofrecer manejo de transacciones declarativo sin utilizar un contenedor EJB, proporciona un mayor control de las excepciones y se integra perfectamente con una gran multitud de librerías entre ellas Hibernate.

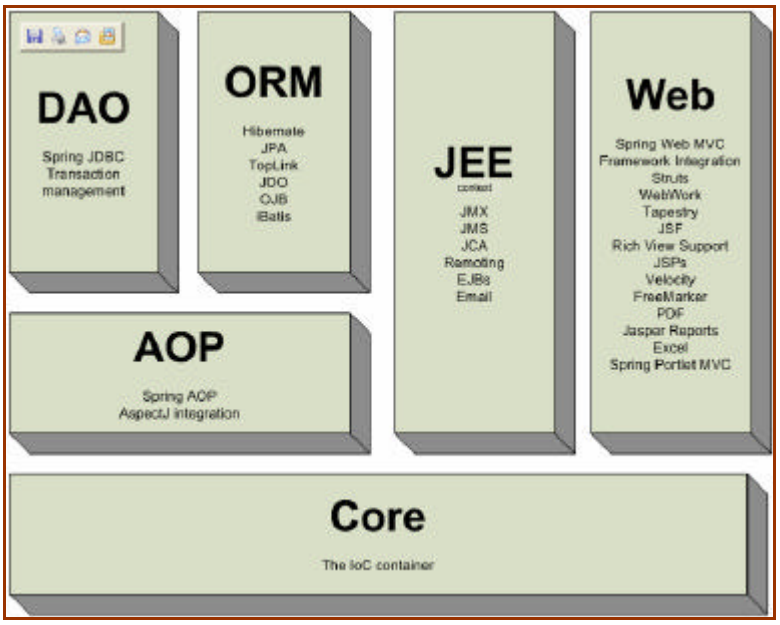
También es importante mencionar que Spring es una herramienta muy probada y con una gran comunidad detrás (cada pocas semanas se presenta una nueva versión).

Una de las metas a conseguir es separar los accesos a datos y los aspectos relacionados con las transacciones, para permitir objetos de la capa de negocio reutilizables que no dependan de ninguna estrategia de acceso a datos o transacciones.

8.2.2.2 Conceptos Básicos

Módulos (Paquetes)

Spring contiene muchas características que le dan una funcionalidad muy amplia, dichas características están organizadas en un conjunto de grandes módulos como se puede observar en el siguiente diagrama:



Esta sección comenta someramente las características de cada módulo:

- *Core o "Núcleo"*, este módulo es la parte fundamental del framework ya que ofrece toda la funcionalidad de Inyección de Dependencias permitiéndote administrar la funcionalidad del contenedor de beans. El concepto básico de este módulo es el BeanFactory, que implementa el patrón de diseño Factory (fábrica) eliminando la necesidad de crear singletons programáticamente permitiéndote desligar la configuración y especificación de las dependencias de tu lógica de programación.

- *Context*, este módulo provee de herramientas para acceder a los beans de una manera elegante, similar a un registro JNDI. El paquete de contexto hereda sus características del paquete de beans y añade soporte para mensajería de texto, como son resource bundles (para internacionalización), propagación de eventos, carga de recursos y creación transparente de contextos para contenedores (como el contenedor de servlets, por ejemplo).

- *DAO*, este módulo crea una capa de abstracción de JDBC que elimina la necesidad de teclear código JDBC tedioso y redundante así como la codificación de códigos de error específicos de cada proveedor de base de datos. También, el paquete JDBC provee de una manera de administrar transacciones tanto declarativas como programáticas, no solo para clases que implementen interfaces especiales, pero para todos tus POJOs.

- ORM, este módulo provee capas de integración para APIs de mapeo objeto - relacional, incluyendo, JDO, Hibernate e iBatis. Usando el paquete ORM se pueden usar esos mapeadores en conjunto con otras características que ofrece Spring, como la administración de transacciones mencionada antes.

- AOP, este módulo ofrece una implementación de programación orientada a aspectos (ayudan a especificar y aislar requisitos de un sistema software, llamados aspectos, que no pertenecen a ningún módulo en particular, sino que pueden afectar a diversas partes del sistema) compatible con AOP Alliance (proyecto estandarizador de lenguaje AOP), permitiendo definir interceptores (patrón interceptor, objetos que se activan en determinadas circunstancias) y pointcuts (lugares dentro del flujo del programa donde se debe disparar el uso del aspecto) de métodos para desacoplar el código de una manera limpia implementando funcionalidad que por cuestiones de lógica y claridad debería estar separada. Usando metadatos a nivel de código fuente se pueden incorporar diversos tipos de información y comportamiento al código, un poco similar a los atributos de .NET.

  La última versión de Spring ha potenciado mucho este concepto, aumentando considerablemente sus posibilidades y simplificando su uso mediante su configuración a través de esquemas Xml.

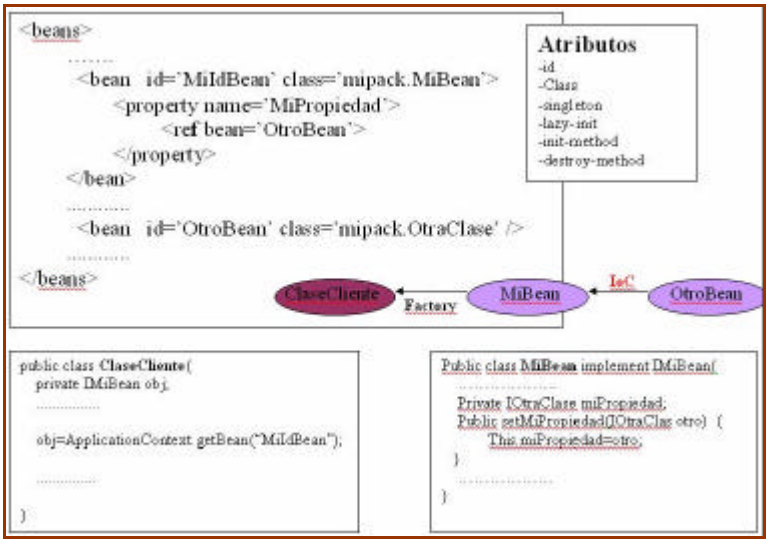
- Web, este módulo provee características básicas de integración orientado a la web, como funcionalidad multipartes (para realizar la carga de archivos), inicialización de contextos mediante servlet listeners y un contexto de aplicación orientado a web. Cuando se usa Spring junto con WebWork o Struts, este es el paquete que permite una integración sencilla.

  El paquete Web MVC provee de una implementación Modelo - Vista - Controlador para las aplicaciones web. La implementación de Spring MVC permite una separación entre código de modelo de dominio y las formas web y permite el uso de otras características de Spring Framework como lo es la validación.

Con los módulos descritos es posible usar Spring en una multitud de escenarios, desde applets hasta aplicaciones empresariales complejas usando la funcionalidad de Spring para el manejo transaccional y el framework Web. Pero Spring no obliga a utilizar todo lo que provee, es decir, no se trata de una solución todo o nada. Existen frontends como Webwork, Struts, Tapestry o JSF que permiten integrarse perfectamente a una capa intermedia basada en Spring, permitiéndote usar todas las características transaccionales que Spring ofrece.

Inversión de Control

Este concepto, base del núcleo de Spring, sigue la idea de que no es necesario llamar a un objeto, 'alguien' ya lo llamará por nosotros.



La figura anterior muestra un ejemplo clarificador, la clase MiBean tiene una propiedad llamada miPropiedad del tipo dado por la interfaz IOtraClase, la cual puede ser cargada mediante el método *set* correspondiente. La clase MiBean no tiene ningún conocimiento de cual es la implementación de miPropiedad. Tampoco hay ninguna línea de código java en ninguna otra clase que lo indique, sino que se delega a Spring su carga, definida en su fichero de configuración de beans. Por tanto, IoC nos facilita cambiar una implementación por otra.

IoC nos será especialmente útil en construcción del patrón DAO. Si tenemos varias implementaciones de persistencia, pongamos JBDC pura e IBatis, la capa de negocio debería construirse con independencia de tales implementaciones, constando en ella únicamente las interfaces DAO. Spring se encargaría de decidir cual implementación tenemos que cargar, si Ibatis o JDBC.

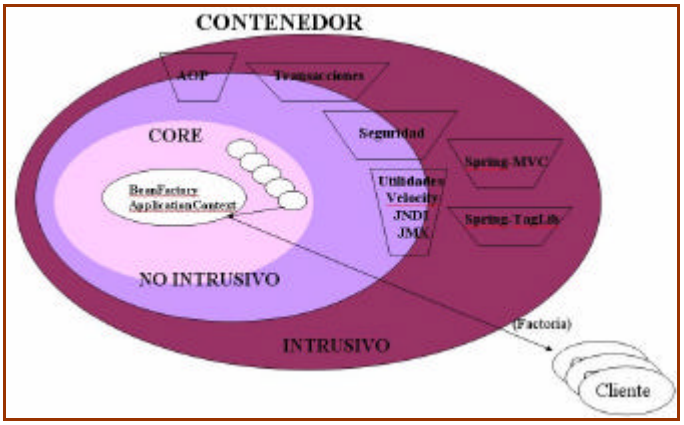
No Intrusivo

Spring es un contenedor de Beans. En principio, toda clase instanciable es susceptible de estar en el contenedor, si bien, tiene el sentido adicional de IoC para aquellas clases que sigan el estándar Java Bean.

En el Core de Spring hay dos clases importantes: BeanFactory y ApplicationContext. La primera representa la factoría de beans y es trasparente al programador de un desarrollo usual. La segunda es usada por los programadores para obtener una instancia del bean en cuestión.

Uno de los objetivos de Spring es ser 'no intrusivo', aquellas aplicaciones configuradas para usar beans mediante Spring no necesitan depender de interfaces o clases propias de Spring, pero obtienen su configuración a través de las propiedades de sus beans. En realidad existe una pequeña dependencia, ya que para invocar a nuestros beans es necesario usar el ApplicationContext propio de Spring. Pero esta pequeña dependencia se puede solucionar envolviendo la instancia ApplicationContext en una factoría propia.

A continuación se muestra una figura que representa esta filosofía:



La gestión de los beans que participan y la manera en que lo hacen, se explicita en ficheros de configuración, xml o properties, que son cargados en el arranque, en la instanciación de la clase ApplicationContext.

Múltiples Mejoras

La última versión de Spring ha ampliado considerablemente su potencia ofreciendo soporte a:

- JPA: nuevo estándar de acceso a base de datos relacionales. Esta funcionalidad sólo esta disponible en Java 5.
- Nueva librería de etiquetas: Para el uso de springMVC , se ha desarrollado una librería de etiquetas muy parecida a la que viene con struts para el tratamiento de formularios. De esta manera hacer una transición del desarrollo con struts a springMVC es más fácil y suave. Es una manera de que la gente se acerque a este completísimo, potentísimo y elegante marco de desarrollo de aplicaciones web, muy superior a struts.
- Contextos: En Spring 1.x la creación de los objetos tenía dos características clave, o eran singleton, y solo había una instancia de ese objeto por aplicación, o era propotype, y cada vez que se necesitan se crea uno nuevo. En Spring 2.0 es posible definir más contextos que controlen la creación de los objetos, aparte de los que Spring traiga por defecto. SpringMVC hace uso de esta característica para por ejemplo controlar que solo se cree un objeto por petición URL en el objeto request, o en sesión. Estos contextos se podrán definir en función de las necesidades de nuestra aplicación.
- Portlets MVC: nuevo marco para el desarrollo de aplicaciones con portlets.
- Lenguajes Dinámicos: posibilidad de cargar una aplicación con spring sin usar java. Es posible definir clases en lenguajes capaces de ejecutarse en la JVM como so BeanShell, JRuby o Grovy. La definición de las propiedades con estos lenguajes se hará por medio de un esquema xml.

Compativilidad

La última versión del framework, Spring 2.0, puede usarse en versiones de java 1.3.1 y superiores. Pero para sacar todo su rendimiento habrá que usar java 5, ya que algunas de sus funcionalidades solo están soportadas por esta especificación.

.

Cualquier aplicación desarrollada contra Spring 1.x puede desplegarse en Spring 2.0 sin problemas

8.2.3 Comparativa

Estos dos frameworks ofrecen un potencia y funcionalidad muy elevada, dando soporte al desarrollo de aplicaciones en las diversas capas que la componen. Realizar una comparación detallada de estas dos herramientas se escapa al objetivo de este documento y podría generar un estudio paralelo muy extenso. Pese a ello incluiré algunas consideraciones interesantes que pueden resultar de ayuda:

- Spring es un framework de aplicaciones Java/J2EE, mientras que EJB es una arquitectura de componentes para el desarrollo y despliegue de aplicaciones empresariales orientadas a objetos y distribuidas.

  Por tanto, Spring y EJB no son una misma cosa: Spring es una implementación y EJB 3.0 es una especificación. Además, EJB soporta el desarrollo de componentes distribuidos, cuestión que no está incluida en la definición de Spring.

- Persistencia Ambos soportan la persistencia de una forma funcionalmente equivalente basada en el mapeo objeto-relacional mediante anotaciones en las clases Java o mediante ficheros XML.

  Spring soporta a la vez varios proveedores de persistencia, como Hibernate, JDO, Toplink, iBatis o cualquier otro proveedor compatible con JPA. EJB soporta cualquier proveedor JPA, que es el API estándar implementado por proveedores como Hibernate, Kodo y Toplink.

  Spring soporta la persistencia JDBC. EJB no.

  Para implementar una caché de objetos compartidos por distintas transacciones, lo que se hace en Spring es fijar la sesión (y la transacción) al hilo de ejecución, usando variables locales del hilo de ejecución. Aunque Spring proporciona clases para facilitar la implementación, en EJB el contexto de persistencia se propaga automáticamente en la transacción.

- Gestión de Transacciones, ofrecen una funcionalidad similar, en ninguno de los dos casos la lógica de negocio tiene que ocuparse explícitamente de la gestión de transacciones, ya que esto se realiza de forma declarativa. En Spring, la demarcación, la propagación y el aislamiento de transacciones se declaran mediante programación orientada aspectos, definiendo proxies para las transacciones en los ficheros de configuración XML. Por otro lado, EJB 3.0 aplica la semántica de transacciones automáticamente en todos los métodos públicos de los beans de sesión, sin requerir configuración adicional. La propagación de las transacciones puede definirse mediante anotaciones o con XML.

  Spring se integra con diferentes APIs de transacciones, incluyendo JDBC, Hibernate y JTA. EJB 3.0 sólo soporta transacciones JTA. Hay que considerar que no todos los contenedores (como Tomcat) soportan JTA, aunque los contenedores EJB sí. En cualquier caso, si se requiere acceder a múltiples recursos en una misma transacción, se requiere una implementación JTA (existen implementaciones open source como JOTM).

- Gestión de Estado, es importante saber que las arquitecturas con estado no tienen el mismo rendimiento ni se escalan igual que las que no tienen estado. Pero hay que tener en cuenta que el estado es importante para muchas aplicaciones, debido a que la interacción del usuario con el sistema implica a veces una serie de pasos con efecto acumulativo.

  EJB proporciona una construcción de primer nivel denominada “Stateful Session Bean (SFSB)” para gestionar el estado tras diferentes llamadas a una misma instancia, que ha sido diseñado teniendo en cuenta cuestiones de escalabilidad y tolerancia a fallos. Estos beans son gestionados por el servidor proporcionando una gestión automática de rendimiento y escalabilidad.

Spring no proporciona ninguna construcción equivalente aunque proporciona otros métodos para obtener el mismo resultado. Para que un bean de Spring tenga estado no debe ser compartido entre llamada. Para ello, el alcance de los beans debe configurarse como “prototype”, de manera que cada vez que se recupera un bean de la factoría se cree una nueva instancia. Por tanto implica una configuración adicional respectoi a EJB.

8.2.4 Conclusión

Es importante destacar que, a priori, no se puede decir que una herramienta sea mejor que la otra. Esto vendrá definido en gran medida por aspectos relacionados con el tipo de aplicación que se quiere construir, su tamaño, el equipo de desarrollo a utilizar, el rendimiento deseado, las previsiones de futuro en temas relacionados con la portabilidad, el mantenimiento, la extensibilidad, la eficiencia...

Aparentemente Spring aporta una mayor flexibilidad que EJB 3 en aspectos relacionados con el desarrollo de la aplicación (persistencia, proveedores de transacciones, compatibilidad tecnológica...), pero como contrapartida nos encontramos con que su utilización y configuración puede resultar algo más compleja que con EJB 3. Debemos evaluar si las diferencias técnicas entre uno y otro son importantes para nuestra aplicación concreta y si el esfuerzo de implementarlas está justificado por su necesidad.

Otro aspecto importante es que, pese a que Spring integra la utilización de muchas tecnologías estándar, él no lo en si mismo. Por otro lado, EJB es una herramienta totalmente estándar. Ésta puede ser una característica importante para la organización que está desarrollando la aplicación y por tanto puede decidir la elección de la herramienta.

De todas formas, aunque se quieran ver como herramientas excluyentes entre si, esto no es cierto. Existen muchas maneras de integrar estas dos tecnologías para obtener los beneficios que pueden aportar cada una de ellas por separado.

8.3 Capa de Persistencia

En esta capa voy a hacer una explicación más breve que en las anteriores ya que considero que el tipo de explicación debe ser más conceptual que técnica.

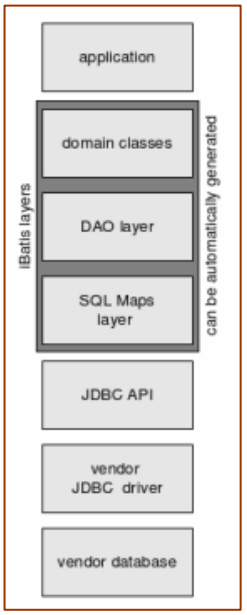
8.3.1 iBatis

8.3.1.1 Introducción

iBatis es un framework de código abierto basado en capas desarrollado por Apache Software Foundation, que se ocupa de la gestión de la capa de Persistencia, facilitando el acceso a los datos del repositorio. Permite que el desarrollador se olvide de la implementación del acceso a datos, únicamente se debe preocupar por realizar una correcta configuración.

Uno de los factores que pueden provocar la necesidad de utilizar una herramienta como esta es,cuando nos encontramos con una aplicación en la que el modelo de datos está creado previamente y no presenta una normalización adecuada. El hecho de que se adapte a este perfil de aplicaciones es debido, básicamente, a que iBatis NO es un ORM (Object Relational Mapper). Además, no es completamente transparente, esto es, el programador manipula el SQL (programa las queries, optimiza las sentencias, etc.).

8.3.1.2 Funcionamiento Básico



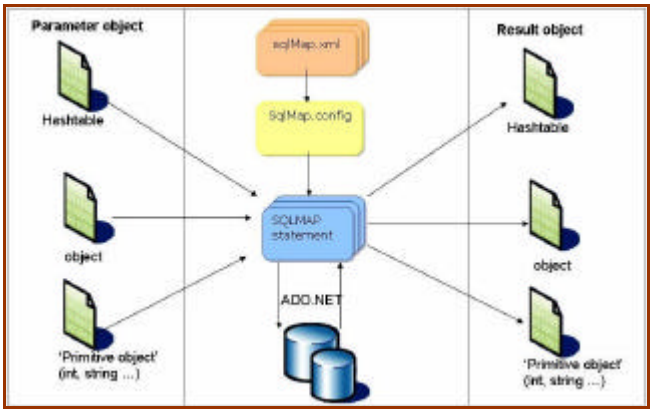
Apache iBatis está constituido por dos frameworks independientes que generalmente se usan juntos: DAO y sqlMaps. El primero simplifica la implementación el patrón de diseño Direct Access Objects (DAO) y el segundo simplifica la persistencia de objetos en bases de datos relacionales:

- *SQL Maps*: iBatis Data Mapper proporciona un modo simple y flexible de mapear los datos entre los objetos y la base de datos relacional. De este modo, se tiene toda la potencia de SQL sin una línea de código JDBC. SQL Maps no sólo reduce considerablemente la cantidad de código necesario para acceder a una base de datos relacional sino que también ofrece una separación entre la capa de negocio y la de acceso a Datos. Este framework mapea la clases a sentencias SQL usando un descriptor XML muy simple.

  Los pasos que se realizan a la hora de utilizar *SQL Maps* son lo siguientes:

  - Crear un fichero de configuración para iBatis. Este es un fichero XML donde se le van a indicar tres cosas basicamente:fichero de propiedades, la fuente de datos JDBC y la declaración de los ficheros de mapeos (SQL Maps).
  - Crear un fichero de propiedades. El fichero de propiedades es un fichero plano que nos simplifica la configuración de iBatis . Los elementos que contiene son: Implementación del Driver JDBC que vamos a utilizar, URL de acceso al servidor, Usuario de acceso a Base de Datos y Password de acceso a la Base de Datos.
  - Crear los ficheros de mapeos correspondientes. Los ficheros de mapeos son ficheros XML que contiene nuestro código SQL para realizar determinadas operaciones con sus parámetros de entrada y salida correspondientes.
  - Crear los diagramas de objetos Java Bean equivalentes. Van a contener los parámetros de entrada y de salida que hemos definido previamente en los ficheros de mapeos. Normalmente estos objetos se corresponderán con la/s tabla/s correspondientes de nuestro Modelo de Datos.
  - Crear el código Java trabajando con los ficheros de mapeos.

A continuación se muestra un diagrama que muestra el flujo de datos:



- *iBatis Data Access Objects (DAO):* Implementación realizada por el framework del patrón DAO. iBatis DAO es una capa de abstracción que oculta los detalles de la capa de persistencia y proporciona un API común para todas las aplicaciones. Con los DAOs se permite configurar una aplicación dinámicamente para usar distintos mecanismos de persistencia.

8.3.1.3 Consideraciones Importantes

Como toda herramienta, no sirve para todo y por tanto conviene tener presente un conjunto de circunstancias que aconsejan su uso y otras en la que no parece tan indicada:

- *SQL Maps*

  Indicado Si:
  - Se requiere una curva de aprendizaje rápida, con pocos conocimientos previos, y evitando aprender lenguajes de consultas como los que usa Hibernate. Evidentemente es necesario SQL.
  - Por el tipo de aplicación o por la estructura de la base de datos se hace necesario el uso directo de sentencias SQL o el acceso a procesos almacenados.
  - Se requiere un alto rendimiento y optimización independientemente de la transparencia.

  No indicado Si
  - Se requiere transparencia y automatización total de la persistencia.
  - Es necesario soporte múltiple de motor de base de datos de forma transparente y automática.

- *DAO es válido cuando:*

  - Se prevé que el motor de base de datos pueda variar en el futuro
  - La implementación del acceso a datos puede variar sustancialmente en el futuro.
  - La implementación del acceso a datos puede variar entre un entorno de producción y otro (software comercial en distintos clientes).

8.3.2 Hibernate

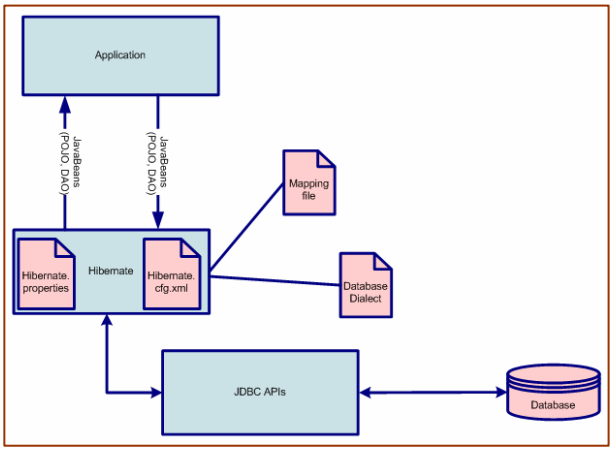
8.3.2.1 Introducción

Hibernate nació hace años de la mano de Gaving King. Gracias a él y a un amplió equipo de trabajo se ha aportado una idea diferente a las existentes para dar soporte a la persistencia de datos en java ha cambiado. Este nuevo enfoque ha planteado un cambio importante en la forma de entender el concepto de persistencia, simplificando en gran medida el trabajo de los desarrolladores y abriendo nuevas vías para el futuro.

Hibernate es una capa de persistencia objeto / relacional y un generador de sentencias sql. Permite diseñar objetos persistentes que podrán incluir polimorfismo, relaciones, colecciones, y un gran número de tipos de datos. De una manera muy rápida y optimizada podremos generar el acceso a bases de datos en cualquiera de los entornos soportados: Oracle, DB2, MySql, etc... Además es open source, lo que supone, entre otras cosas, que no tenemos que pagar nada por adquirirlo.Por tanto estamos ante una herramienta de tipo ORM, esto es, un sistema que permite almacenar objetos de aplicaciones Java en tablas de sistemas de bases de datos relacionales usando metadatos que describen la relación entre los objetos y la base de datos, y lo hace de una manera transparente y autónoma. Comparativa

8.3.2.2 Funcionamiento Básico

En realidad, el funcionamiento conceptual de Hibernate es bastante sencillo, su sencillez depende en gran medida la calidad del diseño de la propia base de datos, un mal diseño puede complicar mucho el uso de Hibernate. Para usar Hibernate se han de crear los objetos de negocio y sus relaciones. Después se deberá crear una serie de archivos xml que indican a Hibernate qué objeto se guarda en qué tabla, indicando la relación entre propiedades del objeto y columnas de la tabla. A continuación se muestra un diagrama de funcionamiento:

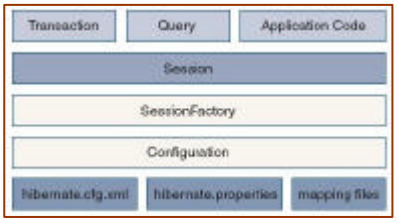


Estos archivos de configuración son:

- El archivo de propiedades de Hibernate (Hibernate.properties) es el encargado de determinar los aspectos relacionados con el gestor de bases de datos y las conexiones con él.
- El fichero hibernate.cfg.xml define la configuración general de toda la herramienta. Pese a que la existencia del fichero anterior (properties), este puede no ser necesario, ya que en éste XML todo lo que se puede con el de propiedades.
- Los archivos que definen el emparejamiento (mapping) de propiedades con tablas y columnas (*.hbm.xml)

De todas maneras las cosas no son tan sencillas con Hibernate como parecen. Hay que tener en cuenta de que Hibernate guarda los datos del objeto en la sesión, y que hasta que no se hace un 'flush' no se persisten realmente los datos. El hecho de que la sesión se interponga entre los datos y la base de datos obliga a tener mucho cuidado en como se gestionan las sesiones. Puesto que una sesión debe pertenecer a un único Thread y un único Thread no debería de tener más de una sesión porque en caso contrario no se puede asegurar la integridad de los datos (dos threads usando la misma sesión podrían estar poniendo en peligro los datos). El siguiente diagrama muestra los diferentes componentes:

8.3.2.3 Consideraciones Importantes



Además de os temas anteriormente comentados. a continuación se enumeran otros factores que hacen que Hibernate sea una herramienta a tener en cuenta:

- *Funcionamiento*. Hibernate ha sido probado en numerosas situaciones, es un proyecto con más de cinco años de antigüedad, con una constante evolución, recientemente tutelado por Jboss group, lo que le augura un largo y prometedor camino.
- *Código Abierto*. A diferencia de otros como TopLink, el pionero en este campo, Hibernate es gratuito Además, si se dispone de la suficiente habilidad, puede extenderse y modificarse según las necesidades. No hay que pagar licencia, no hay limitación al número de conexiones, y su comunidad es inmensa y siempre dispuesta a dar ayuda a quien lo necesite.
- *Documentación* es excelente. Probablemente sea el proyecto de código abierto mejor documentado que haya.
- *Herramientas*. Hibernate proporciona de manera gratuita herramientas que ayudan en toda la fase de desarrollo y producción, haciendo mas fácil tanto la creación de los ficheros, como de las bases de datos asociadas a estos ficheros, o al revés, de objetos asociados a una base de datos existente.
- *Productividad*. Con Hibernate el tiempo de desarrollo de aplicaciones web se reduce considerablemente, al eliminar la necesidad de crear una capa de acceso por nosotros mismos llena de código jdbc. Un solo DAO podría ser capaz de realizar todas las inserciones del sistema.
- *Mantenimiento*. Si algo reduce Hibernate, son las líneas de código que se necesitan en la aplicación, y es más fácil encontrar los problemas. Un cambio en el nombre de una tabla se solucionaría cambiando una palabra en un fichero de configuración. Con Jdbc u otros marcos de desarrollo que usan SQL no se puede decir lo mismo.

8.3.3 Conclusión

La diferencia más importante entre Hibernate e IBatis radica del hecho de que el último basa su funcionamiento en el mapeo de sentencias SQL que se incluyen en ficheros XML. Eso significa que, al contrario que Hibernate, requiere conocimiento de SQL por parte del programador. Por otra parte, permite la optimización de las consultas, ya sea con lenguaje estándar o con SQL propietario del motor de base de datos utilizado. Con iBatis , siempre se sabe lo que se está ejecutando en la base de datos, y tiene herramientas para generar consultas dinámicas muy potentes.Cuando el modelo de datos es muy cambiante o es preexistente al desarrollo de la aplicación (y compartido con otras), iBatis es un claro candidato para ser utilizado. También lo es cuando las relaciones entre las entidades del modelo son muy complicadas, porque con algo de trabajo se puede conseguir que el número de consultas que se pasan a la base de datos no sea excesivo, sobre todo en los listados descriptivos.

iBatis ha ganado peso en la comunidad, hasta llegar a incorporarse al proyecto Apache, y su autor ha publicado un libro monográfico del producto en la serie “in action” de Manning.Uno de los puntos fuertes de iBatis es la estabilidad y la facilidad para encontrar dónde está el problema. Las transacciones y las cachés funcionan sin dar dolores de cabeza.

Por otro lado Hibernate nos ofrece una capa de abstracción total de la base de datos, no necesitamos crear consultas SQL cautivas de una única base de datos, lo cual aumenta considerablemente la portabilidad, mantenimiento y facilidad de uso. Evidentemente a costa de aprender la sintaxis propia de la herramienta y de dedicar un poco de tiempo a entender la forma de implementarlo.

La elección dependerá del tipo de aplicación que necesitemos implementar, así como de su tamaño y de tipo de acceso a datos que necesitemos. Como siempre no existe una única solución.

8.3.4 Comparativa entre Hibernate y EJB 3.0

Pese a que inicialmente no se iba a tratar EJB en la capa de persistencia, existe una comparativa muy interesante que he creído importante incluir por razones didácticas de ambos frameworks:

http://www.adictosaltrabajo.com/t utoriales/tutoriales.php?pagina=hibernateVSEJB3

9 INTEGRACIÓN JSF – SPRING – HIBERNATE

9.1 Introducción

Después de realizar una explicación sobre algunos de los conceptos más importantes que rodean el desarrollo de una aplicación como patrones, arquitecturas, especificaciones, framework, parece claro que la mejor forma de ver su funcionamiento es implementando una pequeña aplicación en la que se integren todos estos conceptos y tecnologías.Por tanto, el objetivo del presente documento, a partir de este punto, es realizar una aplicación, a modo de ejemplo, que muestre como se unen todos estos conceptos.

Se pretende implementar una aplicación web J2EE que trabajara con una BD aun sin decidir y que va ha consistir en un control de Stocks en entorno Web. Por tanto incluirá procesos asociados con el mantenimiento de entidades maestras (productos, departamentos...), además de un conjunto de acciones a realizar (control del stock, inventarios...) .Esta aplicación podría considerarse como un primer módulo de una aplicación mayor destinada a la gestión interna de una empresa a través de un Intranet. Cualquier usuario, dado de alta en la aplicacion podra acceder a ella, prévia autentificación y realizar las tareas propias de la aplicacion.

9.2 Justificación Arquitectónica

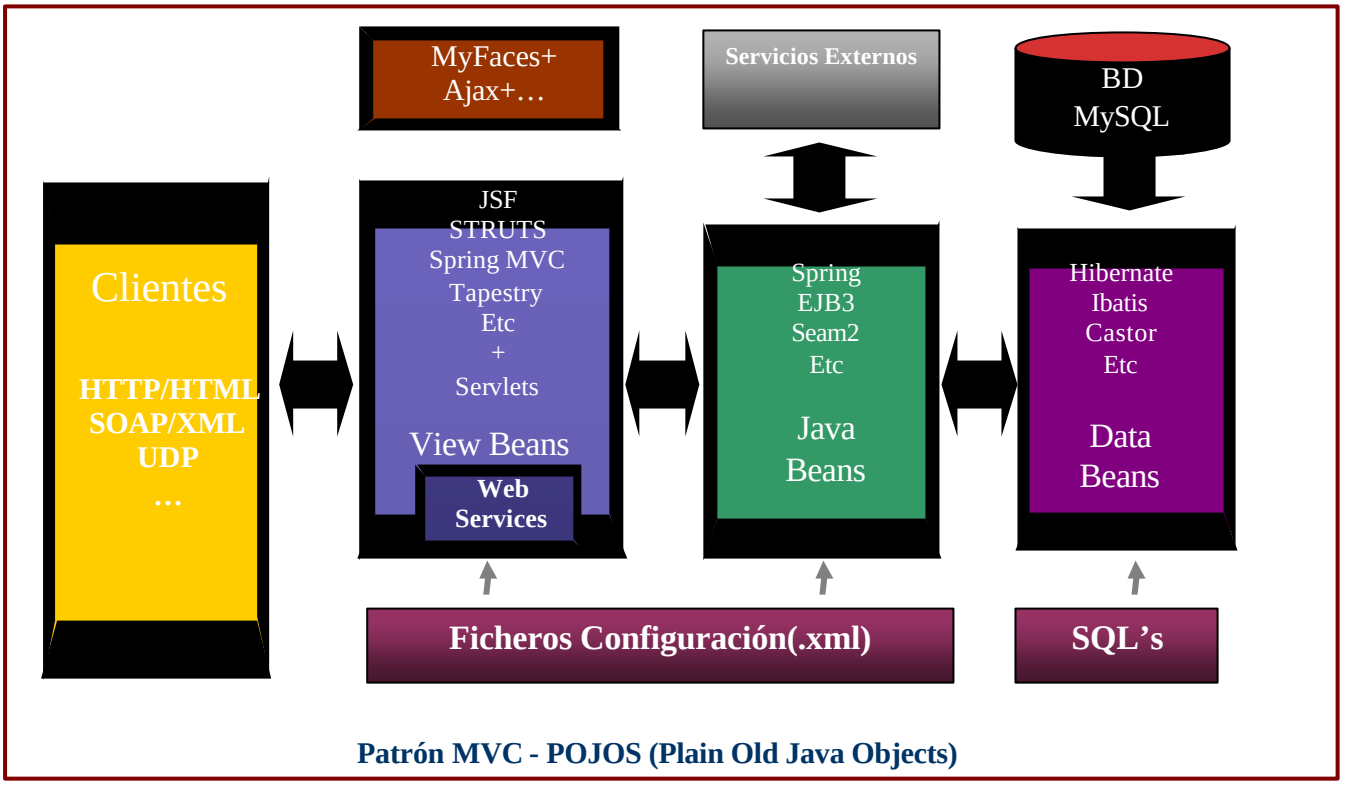
Si bien es cierto el objetivo de toda aplicación es "solucionar problemas de negocio", debemos tomar en cuenta que dicha aplicación debe ser mantenible, basado en estándares como Patrones de Diseño, que se convierten en indispensables en el momento de desarrollar cualquier tipo de Aplicación. Dentro de las Aplicaciones Web podemos citar al Modelo Vista Controlador versión 2 (MVC2) como punto de partida inicial válido, pero para llegar a tener dicho patrón podemos optar por diferentes Framworks, uno de los mas usados incluso por los IDEs mas sofisticados es STRUTS, podríamos usarlo junto con otro Frameworks como SPRING, EJB3, SEAM2…, es decir usar lo bueno de cada uno para lograr un desarrollo mucho mas profesional. Una combinación podría ser STRUTS para la parte de presentación y SPRING para la parte de modelo de negocio, otra combinación podría ser, teniendo en cuenta el gran auge que esta tomado, es el uso de Java Server Faces, utilizando combinaciones del tipo JSF + STRUTS , JSF + SPRING + HIBERNATE, etc. Evidentemente el abanico de posibilidades es muy amplio y por tanto dificulta la elección.

El principal objetivo de una arquitectura es separar, de la forma más limpia posible, las distintas capas de desarrollo, permitiendo un modelo de domino limpio, facilitando las tareas de mantenimiento y permitiendo una correcta evolución de las aplicaciones. Por otro lado es importante que nos facilite el despliegue de la aplicación y el empleo de las tecnologías más adecuadas acorde con el desarrollo que se va ha realizar. Por otro lado, la aplicación que se va a desarrollar pretender dar unas pinceladas sobre una posible arquitectura diseñada para una aplicación del mundo real donde se involucren diferentes frameworks. Para simplificar el ejemplo se ha decidido usar un único frameworks para implementar cada una de las capas, por tanto la labor se centra en la elección de cada uno de ellos.

Por este motivo se ha decidido basar la aplicación en una arquitectura de capas, por lo que admite su funcionamiento tanto en Internet como en Intranet y ofrece la flexibilidad necesaria para ejecutarse en un cliente ligero (navegador web, Wap) o un cliente pesado (Swing, SWT, etc). En este sentido se quiere evitar la reescritura de código y permitir la reutilización de las capas en cada.

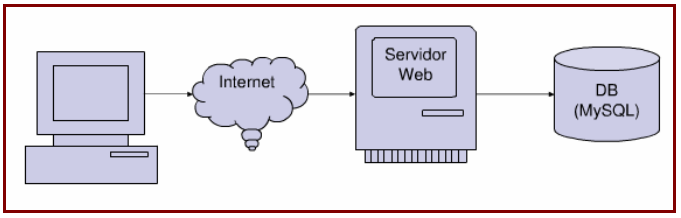
La aplicación utiliza una arquitectura multi-capa no-distribuida. La figura siguiente muestra el particionamiento de las capas de la aplicación y las tecnologías elegidas para cada capa. También sirve como diagrama de ejemplo de despliegue de la aplicación. Para una arquitectura colocada, las capas de presentación, de lógica-de-negocio y de integración están físicamente localizadas en el mismo contenedor Web. Interfaces bien-definidos aíslan las responsabilidades de cada capa. La arquitectura colocada hace que la aplicación sea más simple y escalable.

Se ha decidido, por simplicidad definir únicamente tres capas tal y como se muestra en la siguiente figura:



Patrón MVC - POJOS (Plain Old Java Objects)

Distribución física del sistema:



9.3 Justificación Frameworks

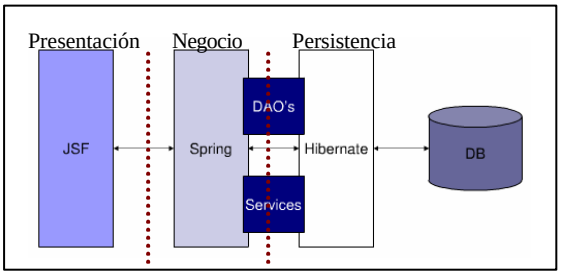
Para la capa de presentación (la vista) se quiere el framework que proporcione mayor facilidad en la elaboración de pantallas, mapeo entre los formularios y sus clases en el servidor, la validación, conversión, gestión de errores, traducción a diferentes idiomas y, a ser posible, que facilite la inclusión de componentes complejos (menús, árboles, Ajax, etc) de forma sencilla y, sobre todo, fácil de mantener. Por ello, para esta capa se ha elegido una implementación de JSF, en concreto MyFaces ya que aporta un conjunto de elementos suplementarios a JSF aumentando considerablemente las posibilidades de la aplicación. MVC es el patrón de diseño arquitectural recomendado para aplicaciones interactivas Java. MVC separa los conceptos de diseño, y por lo tanto decrementa la duplicación de código, el centralizamiento del control y hace que la aplicación sea más extensible. MVC también ayuda a los desarrolladores con diferentes habiliades a enfocarse en sus habilidades principales y a colaborar a través de interfaces claramente definidos. MVC es el patrón de diseño arquitectural para la capa de presentación. JSF encaja bien en la arquitectura de la capa de presentación basada en MVC. Ofrece una clara separación entre el comportamiento y la presentación. Une los familiares componentes UI con los conceptos de la capa-Web sin limitarnos a una tecnología de script o lenguaje de marcas particular.

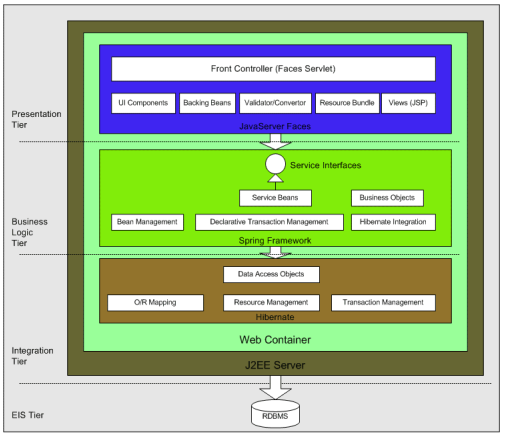
Para la capa de negocio se ha optado por una solución basada en servicios (no necesariamente servicios web, aunque permitiendo su integración de forma limpia) que trabajaban contra un modelo de dominio limpio. La persistencia de las clases se sustenta en DAO's (Objetos de Acceso a Datos), manteniendo aislada la capa de persistencia de la capa de negocio. Tanto los servicios como los DAO's así como el propio modelo son realmente POJOs (clases simples de Java), con la simplicidad que conllevan y sin dependencias reales con ningún framework concreto. Para realizar esta integración se ha elegido Spring. Como se verá más adelante, para la implementación de la aplicación se utilizan algunos patrones conocidos como Singleton, Fachada…

Para la capa de persistencia se ha decidido utilizar alguna herramienta ya existente, que permitiese realizar el mapeo objeto-relacional de una forma cómoda pero potente, sin tener que implementarlo directamente mediante JDBC. Esto último conllevaría, por ejemplo, un esfuerzo importante en un caso de cambio de base de datos, en la gestión de la caché, etc. Además se quiere evitar la escritura de sentencias SQL. La herramienta elegida finalmente es Hibernate.

Quedará pendiente incluir el soporte a temas de seguridad. Esta aplicación de ejemplo no incluye ésta capa de seguridad, aunque en caso de hacerlo seria conveniente seleccionar una que fuese totalmente independiente de cada contenedor concreto, común a todos ellos y que ofreciese una gran versatilidad.

Además, un motivo común para la elección de cada uno de los frameworks es, aparte de su potencia y especialización en la capa para la que se han seleccionado, su impacto en la comunidad, la calidad de los desarrollos realizados, la madurez de la herramienta y la cantidad y calidad de la documentación existente. Así pues la configuración resultante queda definida en las siguientes figuras:





Además de los frameworks sobre los que se va ha centrar el proyecto no hay que olvidar que existen muchas otras tecnologías que combinadas enriquecen el contenido Web y mejoran la experiencia de usuario. Con esto se quiere dar su importancia a las tecnologías que forman parte de la tendencia WEB 2.0. y que se han utilizado en el diseño de la aplicación: entre ellas destacamos el uso AJAX. Estas tecnologías son conocidas como RIA (Rich Internet Aplications) y su objetivo es el de proveer a los sitios Web de capacidades multimedia así como mayor una mayor eficiencia evitando las cargas constantes de contenido. Dentro del proyecto se ha utilizado tecnología AJAX para hacer la carga de algunos de los selectores de tipo lista. Gracias a esta técnica cuando se selecciona una opción de la lista, no necesita cargarse de nuevo para mostrar su contenido, sino que se realiza de manera asíncrona, además con la selección se obtiene el objeto representado por la cadena.

9.4 Beneficios Aportados

La primera y principal ventaja de la solución presentada se deriva de la modularidad del diseño. Cada una de las partes empleadas (JSF -MyFaces para la vista, Spring para la integración, Hibernate para la persistencia) es intercambiable de forma sencilla y limpia por otras soluciones disponibles. Por ejemplo, para la vista se emplea Java- Server Faces, pero nada impide emplear también una aplicación de escritorio mediante Swing o SWT sin tener que tocar ni una sola línea de código de las restantes capas. Es más, nada impediría que se pudiese disponer de una aplicación con una parte de la capa de presentación en JSF y otra parte, para otro tipo de usuarios, en Swing, ambas funcionando a la vez y compartiendo todo el resto del código (lógica de negocio, persistencia, integración, seguridad, etc).

De igual forma, si se desean cambiar elementos de la capa de persistencia empleando otro framework para el mapeo diferente de Hibernate, o sencillamente no utilizar ninguno, tan sólo serían necesarios cambios en esa capa. De la misma manera se podrían sustituir cualquiera de las otras capas. El diseño se ha hecho reduciendo al mínimo posible las dependencias entre ellas.

9.5 Elección del Gestor de Datos

La herramienta seleccionada para dar soporte a la persistencia de datos es MySql.Para justificar la elección de esta herramienta frente a otras posible he tenido en cuenta las siguientes ventajas aportadas por la herramienta seleccionada:

- MySql es open-source, y no hay que pagar licencia. Esta es una de las razones por la que es ofrecido en el servicio básico de casi todos los hostings. En esta línea tenemos a otro posible candidato PostgreSQL, los siguientes puntos justificarán la elección final respecto a este último.
- Porque MySql tiene un release para windows bastante simple de instalar y usar (como servicio en los windows de la familia NT) y PostgreSQL no.
- Porque es rápido y fácil de usar. Entendiendo por fácil, por ejemplo, que no hay que perder mucho tiempo investigando, en una documentación extensa, para averiguar a que descripción se corresponde un determinado número de error poco explicativo (ORACLE). Además al ser open-source la comunidad existente detrás del desarrollo y que utiliza el programa es mucho más activa y se detectan y resuelven los problemas más velozmente. La comunidad Postgre es mucho menor.
- Porque MySql se adapta perfectamente al entorno del proyecto ya que se integra perfectamente con Hibernate en un entorno web, ofreciendo una base de datos simple, económica y potable. Es una base de datos muy confiable y estable, y no da problemas, aún con bases de datos gigantescas (más de 10Gb), y con muchos usuarios simultáneos.

10 APLICACION EJEMPLO

10.1 Introducción

La aplicación que se presenta consiste en un primer módulo de una aplicación destinada a la gestión interna de los procesos de una empresa. Este módulo consiste en un control de Stocks. La funcionalidad implementada es, únicamente una pequeña parte de las posibilidades que ofrece una aplicación de este estilo, en un futuro se le podrían añadir nuevos módulos como por ejemplo, la facturación, los cobros, contabilidad.... El objetivo del presente ejemplo es ver como se pueden integrar las diferentes tecnologías seleccionadas en una única aplicación.

La aplicación esta orientada a una empresa de venta de componentes y servicios orientados al sector de la logística y el trasporte. Sistemas de control gestión y localización de flotas de vehículos y de personas. Por tanto el uso de esta aplicación lo realizará personal experimentado. Además la variabilidad de materiales no es muy elevada, pero si sus movimientos.

10.2 Requerimientos funcionales de la aplicación

La aplicación de ejemplo que se va ha construir, pretende ser una aplicación aplicable en el mundo real y que por tanto sirve como perfecto marco de referencia para aplicar las tecnologías anteriormente comentadas.

La primera fase en el diseño de una aplicación Web es decidir y detallar los requerimientos funcionales del sistema. En nuestro caso nos encontramos ante una aplicación de control de Stocks. Se utiliza un análisis de casos de uso para definir los requerimientos funcionales de la aplicación:

Este diagrama identifica los actores del sistema y las operaciones que pueden realizar. En la aplicación de ejemplo se implementarán nueve casos de uso, que son los que aparecen en amarillo. El resto de casos quedan pendientes de implementar, los que están en gris. Pese a ello, la aplicación contempla las entidades asociadas a los casos de uso en gris por coherencia de datos, pese a que no ofrece ninguna funcionalidad para su tratamiento. Esto quiere decir que la aplicación contemplará, por ejemplo, la existencia de la entidad empresa, aunque no se implementará la parte asociada a su gestión directa.Tenemos una aplicación multiempresa, donde cada empresa puede tener diversos centros, con usuarios que acceden al control de Stock. Las acciones que debe permitir realizar la aplicación son:

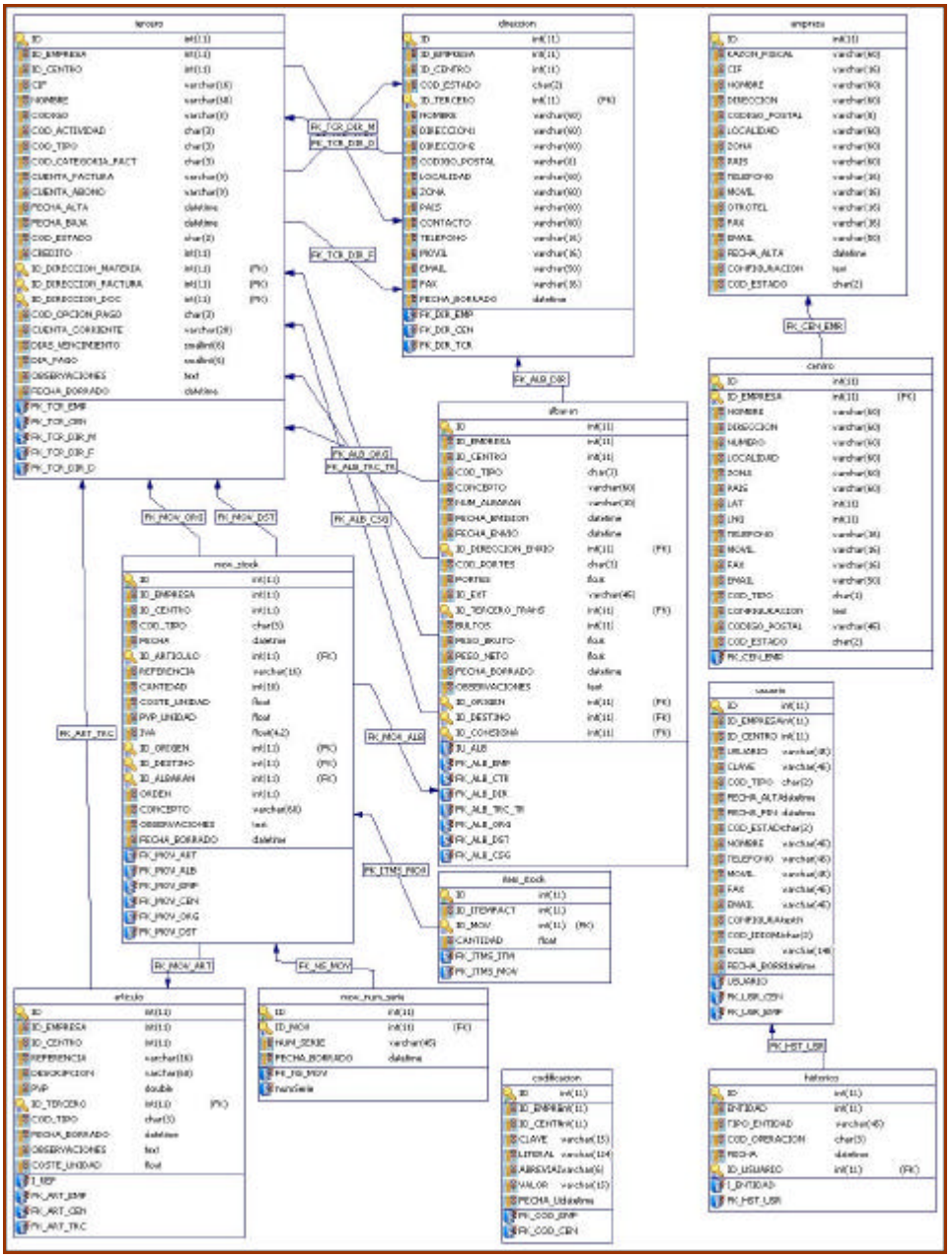
1. Gestión Empresa: Comprende los datos asociados a la empresa. Además da soporte a implementar la aplicación multiempresa. Incluye la visualización de los datos, así como su alta, baja o modificación.
2. Gestión Centros: Comprende las diferentes delegaciones o centros que pueden componer la empresa. Incluye la visualización de los datos, así como su alta, baja o modificación.
3. Gestión Usuarios: Datos asociados a los usuarios que acceden a la aplicación. Incluye la visualización de los datos, así como su alta, baja o modificación.
4. Codificación: Datos auxiliares asociados a diferentes conceptos como por ejemplo contadores de las entidades maestras de la aplicación, datos para los diferentes desplegables de la aplicación. Incluye la visualización de los datos, así como su alta, baja o modificación.
5. Logín: Proceso en el que un usuario se identifica y recibe los permisos de acceso correspondientes.
6. Gestión de Artículos: Datos generales de los diferentes productos que la empresa puede adquirir. Se contemplan artículos de tipo granel o asociados a un determinado número de serie. Incluye la visualización de los datos, así como su alta, baja o modificación.

Los casos de uso comentados hasta ahora no se implementaran en la aplicación de ejemplo. Pero si se ha previsto su inclusión futura incorporando los objetos y datos necesarios para acceder de forma estática.

7. Gestión de Proveedores: Gestiona las diferentes empresas que suministran los artículos para los que se realiza el control del Stock. Dentro de este caso de uso se incluye la visualización de los datos asociados a los diferentes proveedores, así como su alta, baja o modificación.
8. Gestión de Clientes: Entendemos por cliente, las diferentes empresas a la que se suministran los artículos para los que se realiza el control del Stock. Se incluye la visualización de los datos asociados a los diferentes clientes, así como su alta, baja o modificación.
9. Gestión de Almacenes: Gestiona las diferentes localizaciones en las que podemos tener los artículos almacenados. Un almacén pude corresponderse a una ubicación física o más lógica, un ejemplo de este último podría ser el almacén de alquiler, de reparación, de sustitución.... Se incluye la visualización de los datos asociados a los diferentes almacenes, así como su alta, baja o modificación.
10. Gestión de Artículos: Visualización, alta, baja o modificación de datos de artículos.
11. Entrada de Material: Incluye todas las acciones asociadas a la recepción de artículos por parte de proveedores o clientes. Esta acción da lugar a la generación de un documento albarán, que posteriormente puede servir para diversas cosas, como por ejemplo la realización de inventario.
12. Salida Material: Incluye todas las acciones asociadas a la venta de artículos a los diferentes clientes. Esta acción da lugar a la generación de un documento albarán,.
13. Movimientos: Incluye los diferentes movimiento que puede sufrir un determinado artículo siempre que no se corresponden con la recepción de proveedor o la venta a cliente. Puede, por ejemplo tratarse de movimiento entre los diferentes almacenes de la empresa. Cada movimiento da lugar a un albarán.
14. Inventario: A partir de la historia de entradas y salidas de material muestra la disponibilidad de artículos en un determinado momento.
15. Números de Serie: Un número de serie es un código asociado a un artículo. Pretende mostrar la historia de un determinado número de serie a partir de sus entradas, movimiento y salidas.
16. Histórico: Es un control de las acciones realizadas por los usuarios. Guarda información de seguimiento respecto a altas, bajas o actualizaciones.
17. Gestión de Direcciones: Pretende ser una gestión centralizada de las posibles direcciones que puede tener un cliente, proveedor.... Además, estas direcciones se pueden crear o modificar en el momento de realizar algún movimiento de artículos.

10.3 Diseño de la base de datos

A continuación se muestra el esquema de base de datos de la aplicación. Se han eliminado las claves foráneas hacia la tabla empresa y centro para facilitar la comprensión del esquema.



Es importante citar un conjunto de comentarios al esquema anterior:

1. Todas las entidades tienen un código único que las identifica.
2. La tabla 'tercero' pretende agrupar las entidades cliente, proveedor y almacenes. Los datos asociados a estas entidades se persisten en la misma tabla. Además, esta tabla contiene algunos datos relacionados con un futuro módulo, facturación, que no se implementa en este ejemplo, pero sirven para ejemplarizar la futura evolución de la aplicación.
3. La tabla 'albaran' sirve para realizar una agrupación de un conjunto de entradas, salidas o movimientos de artículos. Esta agrupación servirá para futuros procesos de facturación.
4. La tabla 'direccion' es una unificación de los datos asociados a las direcciones. De esta forma se evita la repetición de información y se facilita y flexibiliza su tratamiento.

5. Las tablas ‘mov_stock’, ‘item_stock’ y ‘mov_num_serie’ definen como se ha realizado una determinada entrada, salida o movimiento de artículos. Cada uno de ellos da lugar al un conjunto de entradas en estas tablas. La primera es la cabecera de la acción y se enlaza directamente con el albarán. La segunda define los artículos asociados (sin número de serie, o sea, por ejemplo artículos de tipo granel). La tercera sirve para separar los artículos asociados que disponen de número de serie.
6. La tabla ‘codificacion’ contiene todos los datos auxiliares necesarios para el funcionamiento de la aplicación. Códigos únicos de la tablas de entidades maestras, datos asociados a listas desplegables ...
7. La tabla ‘historico’ pretende servir de seguimiento para las acciones de modificación, alta y baja realizadas por un usuario sobre las principales entidades de la aplicación.

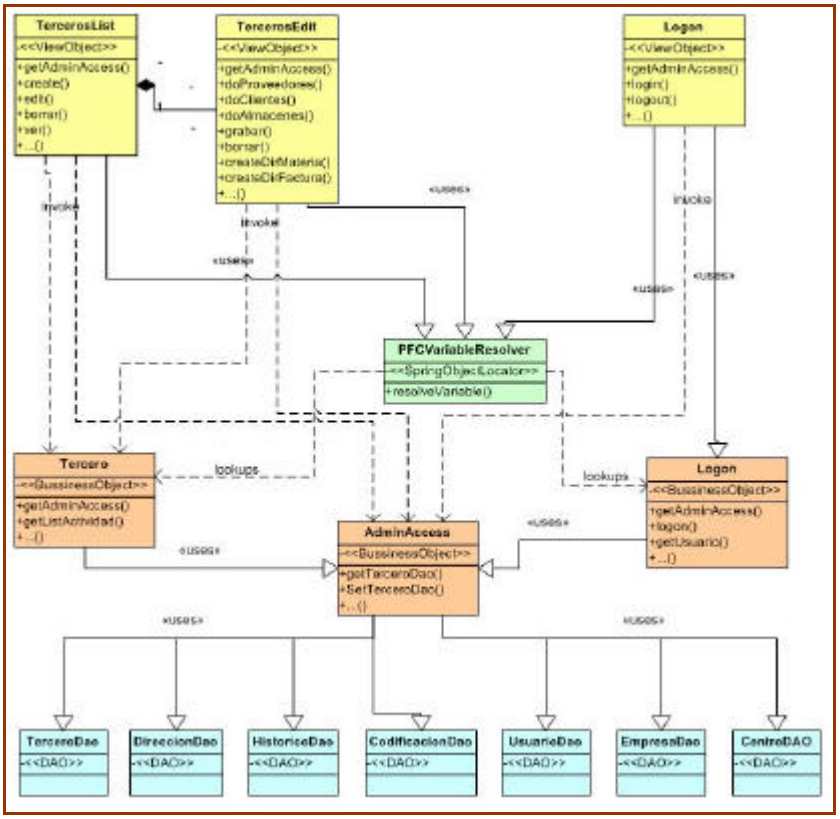
10.4 Caso de Uso: Gestión de Proveedores

Como el objetivo de este proyecto no es el análisis exhaustivo de una aplicación, sino la ejemplificación del proceso de integración de diferentes framewoks para crear una aplicación del ‘mundo real’, sólo se va ha analizar una parte de uno de los casos de uso presentados, entendiéndose que el resto se comportan de forma similar.

En realidad, los casos de uso ‘Gestión de Proveedores’, ‘Gestión de Clientes’ y ‘Gestión de Almacenes’ comparten los mismos objetos en las diferentes capas de la aplicación. Todos ellos se integran dentro del concepto ‘Tercero’. Por tanto, los diagramas que se muestran a continuación son iguales para cualquiera de estos tres casos. . Se ha seleccionado la creación de un tercero para ejemplarizar el proceso

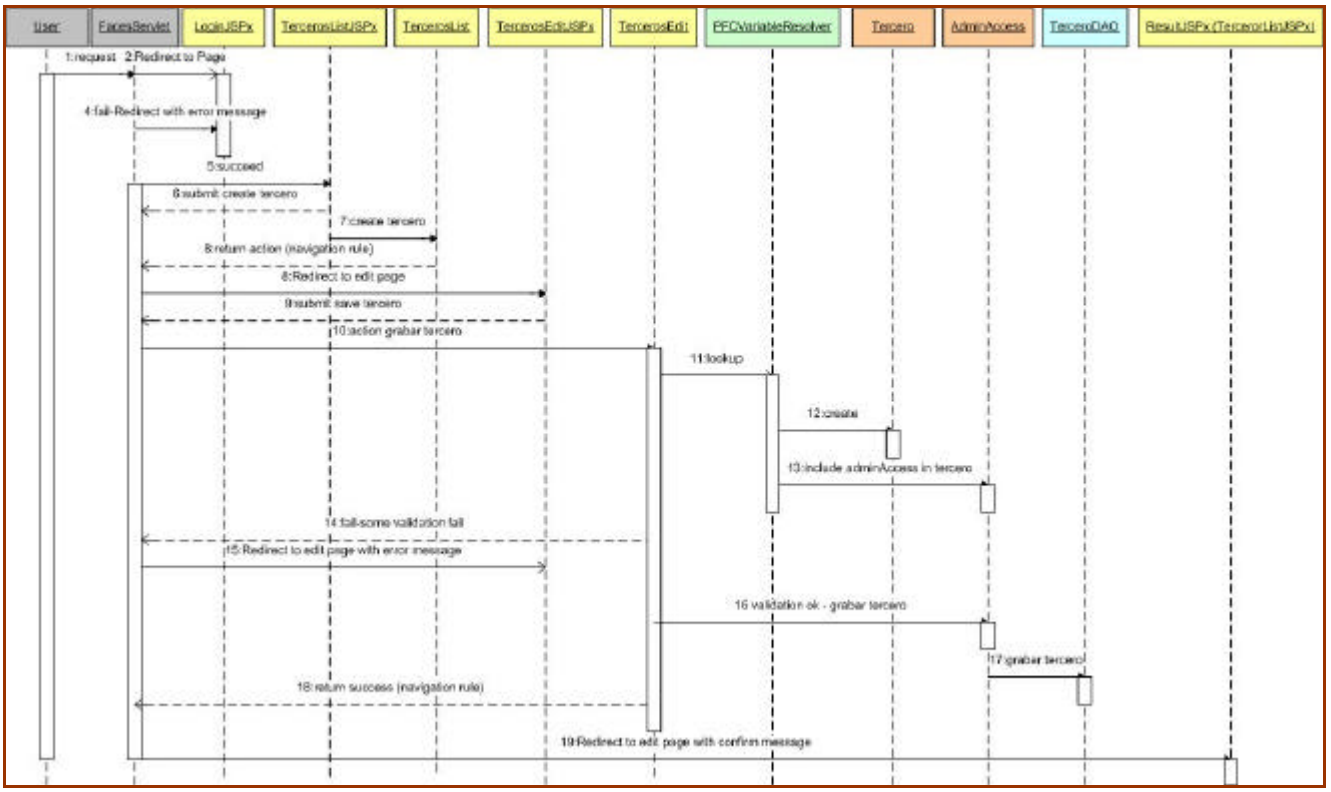
10.4.1 Modelo Conceptual

Al realizar el diseño se ha decidido no usar interfaces (solo algunos básicos) para simplificar la aplicación. En la capa de presentación se usan beans de respaldo (ViewObject). Por debajo se encuentra la capa de negocio, donde destaca un objeto (AdminAccess) que centraliza los accesos a la capa de integración. Ésta contiene los objetos DAO con sus implementaciones en hibernate. El contexto de aplicación Spring conecta y maneja la mayoría de los beans dentro de las diferentes capas. El objeto PFCVariableResolver integra JSF con la capa de lógica de negocio.



10.4.2 Diagrama de Flujo

El siguiente diagrama muestra, de forma simplificada, el flujo de peticiones que se produce en el sistema desde que el usuario accede a la aplicación a través de la página de login hasta que consigue dar de alta una nueva entidad tercero. En el diagrama no se ha contemplado, dentro del alta de un tercero, la parte asociada a la creación de direcciones o a la visualización de los datos históricos, ya que esta parte incluiría la aparición de nuevas entidades de negocio y de integración que harían más complicada la compresión del diagrama.



La implementación de la capa de presentación (parte del diagrama en amarillo) implica crear las páginas jspx, definir las reglas de navegación por las diferentes páginas en función de los resultados de las acciones de usuario, confeccionar los beans de respaldo (view), y realizar la integración de JSP con la capa de negocio a través de Spring.

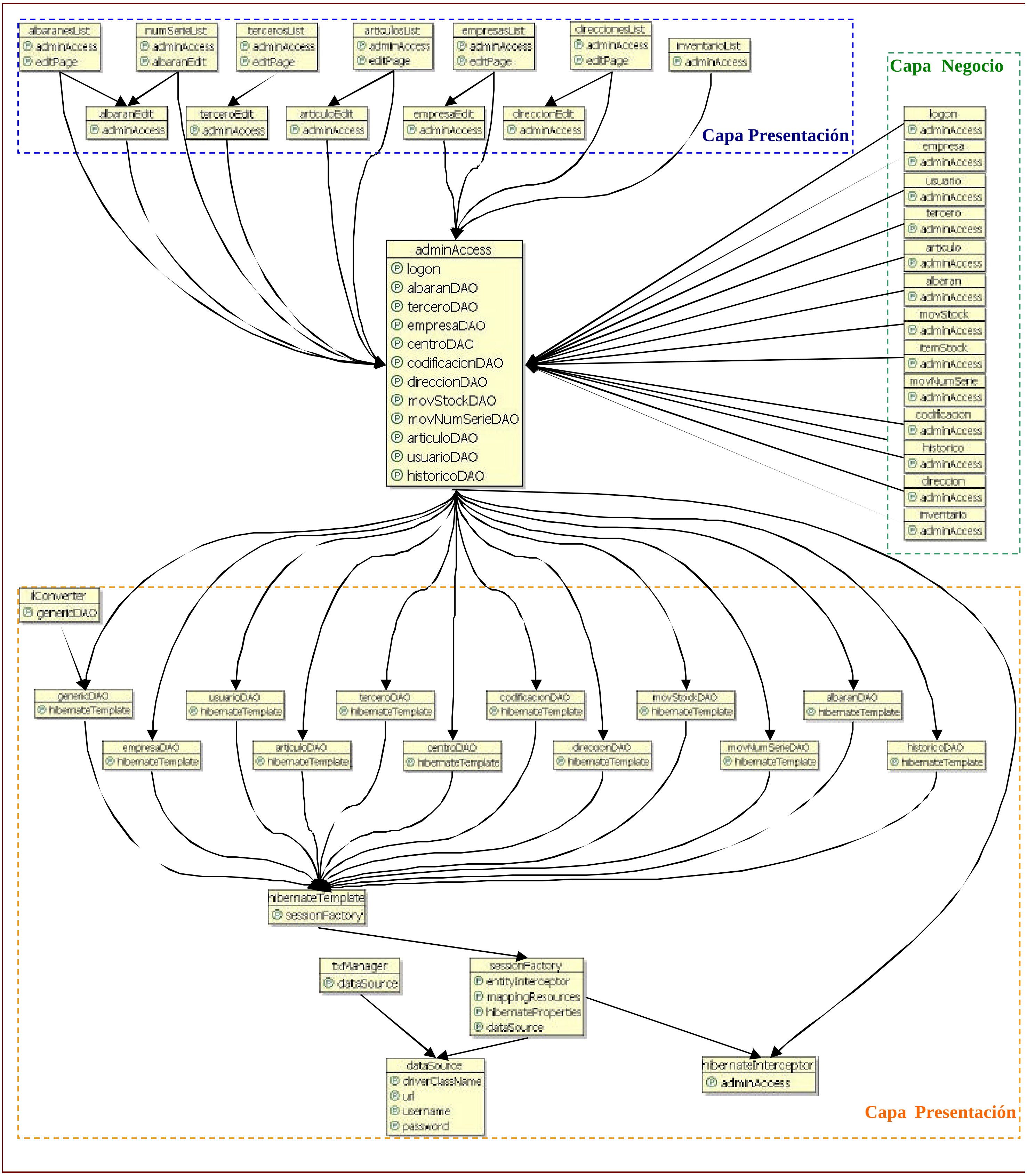
En la capa de negocio (parte del diagrama naranja) se han de definir los objetos de negocio. En la aplicación, se ha tomado la decisión de añadir un objeto (adminAccess) que permite el acceso unificado a la capa de integración. Este objeto contiene todos los objetos DAO del sistema a modo de objetos singleton, es una factoría de objetos DAO para la aplicación, con ello se pretende tener un nexo directo a los datos desde cualquier punto, aunque no se trata de una decisión académicamente correcta, facilita el desarrollo del sistema. Finalmente, en esta capa tenemos la conexión de todos los objetos mediante Spring (IoC).

Por último esta la capa de integración (parte azul del diagrama) donde Hibernate mapea objetos a la base de datos relacional utilizando un fichero de configuración (*.hbm). En el diagrama anterior solo se han mostrado los objetos DAO, pero faltan todos los objetos que hacen que hibernate se pueda comunicar con la base de datos.

Todas las entidades presentadas son explicadas más adelante dentro del presente documento.

10.5 Representación Grafica de los Beans de Spring

A continuación presenta un gráfico con la relación de beans que gestiona Spring dentro de la aplicación. Todos estos objetos están definidos dentro del fichero ApplicationContext.xml que se describe más adelante. El objeto encargado de resolver las peticiones a Spring es PFCVariableResolver.java.



10.6 Paquetes del Proyecto

En este punto del documento se realiza una breve explicación de los objetos más importantes contenidos en cada uno de los paquetes (package) del proyecto.

- *com.pfc.admin.base:* Contiene los objetos básicos de la aplicación y una clase con utilidades.

  - AdminAccess.java: Se trata de un objeto que implementa 'ApplicationContextAware' con lo estamos obligados a implementar:

    ```
    public void setApplicationContext(ApplicationContext arg0) throws BeansException {applicationContext = arg0;}
    ```

    con esto tenemos acceso al ApplicationContext, donde está definido todo el funcionamiento del IoC. Esto nos permite centralizar el acceso a todos los DAO (objetos singleton) desde cualquier entidad (a todas las entidades se les inyecta este objeto). También se centralizan métodos para personalizar la notificación genérica de notificaciones de aviso, mensajes o errores, el acceso a objetos globales como el *logon*, el acceso a métodos para recuperar otros datos genéricos...

  - HibernateInterceptor.java: Hay situaciones en que puede ser necesario realizar algún conjunto de operaciones pre-requisito o post-requisito entre el acceso a la lógica de negocio y el de datos. En tal caso, puede usarse un interceptor para interceptar el proceso de negocio proporcionando una extensión de funcionalidad o una extensión de características. En la aplicación de ejemplo no se hace uso de esta funcionalidad pero se ha incluido para ver sus posibilidades.

  - StdDAO.java: Pretende unificar código para todos los objetos DAO de la aplicación.

  - StdEditPage.java, StdListPage.java: Son clases base para todos comunes para los beans de respaldo de cada una de las páginas. Estos beans están en el paquete '....pages'.

- *com.pfc.admin.beans:* Contiene otros beans necesarios en la aplicación para la gestión del logon o para la lógica asociada a la realización de un proceso de inventario. Además contiene otro objeto que necesita una explicación más detallada:

  - ILConverter.java: Esta clase es la que centraliza el funcionamiento principal de Ajax. Implementa la clase Converter. Esta clase debe implementar, como mínimo, los métodos getAsObject y getAsString. El método getAsObject realiza la conversión del dato de tipo String al valor en el tipo de datos que se desee, en nuestro caso un objeto y el otro realiza el camino inverso. Por ejemplo, a partir del nombre de un tercero recibido desde el control Ajax es capaz de resolver el objeto asociado y viceversa. Para ello se necesita, adicionalmente, que las entidades implemente algunos métodos como 'public String toString()', 'public boolean equals (Object obj)'. Su uso se realiza desde la página *.jspx de la forma:

    ```
    <s:inputSuggestAjax size="40" suggestedItemsMethod="${tercero.adminAccess.terceroDAO.getClientesAjax}"
                charset="utf-8" itemLabelMethod="${ilConverter.getLabel}"
                value="${value}" autoComplete="false"
                maxSuggestedItems="50" converter="${ilConverter}" />
    ```

    Para acabar de completar el ciclo Ajax, falta la fuente de datos que se indica con la propiedad 'suggestedItemsMethod'. Se identifica un objeto DAO como el encargado de recuperar la lista de objetos de la persistencia. En realidad falta definir el comportamiento del Tag de MyFaces 'inputSuggestAjax', pero esto se hará al describir el paquete 'facelets'.

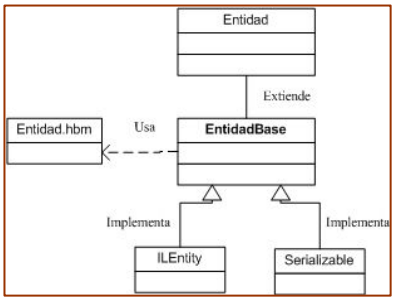
- *com.pfc.admin. facelets:* Contiene únicamente la clase InputSuggestAjaxComponentHandler.java. Ésta se encarga de gestionar el comportamiento del Tag de MyFaces 'inputSuggestAjax'. De esta forma se puede hacer que el componente visual nos proponga un texto, asociado al atributo del objeto indicado, a partir de la cadena que el usuario va insertando en el control. Esta característica es muy útil para facilitar la labor del usuario de la aplicación.

- *com.pfc.admin.entities:*Contiene los ficheros de descripción de mappings (*.hbm)

  Indican cómo enlazar los datos entre una clase Java y una o más tablas de la BBDD. Se definen todas las claves foráneas y su ámbito (uno a uno o muchos a muchos).
  También permiten definir comportamiento en cascada en acciones. Un ejemplo es el borrado en cascada de hijos huérfanos ('all-delete-orphan')
  Una de las características más interesantes es la posibilidad de definir una relación de la tabla con tablas hijo, de manera que hibernate pueda retornar automáticamente registros de otra tabla (etiqueta <bag></bag>). Un ejemplo es poder recuperar las líneas de movimiento de stock ('mov_stock') asociadas a un albarán y la forma en que hibernate debe manejar esta relación. Toda esta gestión la realizará automáticamente hibernate.

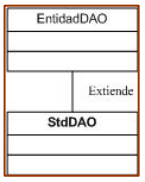
  Contiene las clases (*.java) que definen las reglas de negocio de las entidades. Estas entidades extienden su equivalente en el paquete 'base' que se explica a continuación.

- *com.pfc.admin.entities.base.* Contiene las clases (*.java) que se generan a partir de los ficheros (*.hbm). Para generar estas clases se ha usado el plugging 'Hibernate Synchronizer'. Permite generar automáticamente código de Business object para Hibernate (persistencia). Cuando se realiza un cambio en base de datos, es necesario recrear los *.hbm y el objeto *.java asociado, 'Hibernate Synchronizer' lo hace automáticamente. Con esta capa se separa totalmente la persistencia de las reglas de negocio.



- *com.pfc.admin.interfaces:* Contiene un interfaz común para todas las clases del paquete anterior. Es importante ya que obliga a implementar métodos que todas las clases deben tener.

- *com.pfc.admin.entities.DAO:* La aplicación de ejemplo utiliza el Patrón DAO. Este patrón abstrae y encapsula todos los accesos a la fuente de datos. Cada entidad tiene, en este paquete, un objeto DAO que gestiona todos sus accesos a datos (adicionalmente se ha añadido un DAO genérico para otros accesos, como puede ser temas relacionados con ajax). Estas clases extienden un objeto DAO básico que contiene funcionalidad común (paquete base). No se han usado interfaces para simplificar el diseño.



- *com.pfc.admin.pages:* Son los beans que definen las propiedades y la lógica asociada a los componentes UI usados en cada una de las páginas. Normalmente hay uno por página. Todos extienden una clase para compartir código.

- *org.springframework.web.jsf:* En este paquete encontramos la clase 'PfcVariableResolver.java'. Podemos observar que en el fichero de configuración 'faces-config.xml' se hace referencia a esta clase de la forma:

```
<application>
<!-- Dirige las peticiones de Beans a Spring-->
<variable-resolver>org.springframework.web.jsf.PfcVariableResolver</variable-resolver>
</application>
```

  Esto indica que clase ha de encargarse de resolver los objetos cuando estos sean pedidos. Es en concreto el método 'resolveVariable(...)' el que se encarga de resolver las peticiones de objetos. Lo primero que intenta es obtener el objeto a través de la factoría de objetos de Spring. En caso de no obtener resultados, delega la búsqueda a faces.

- *Carpeta WebContent / components:* Contiene los componentes facelets diseñados para poder ser reutilizados en diferentes páginas.. Para conseguir que las páginas se rendericen con componentes facelets debe incluirse la siguiente entrada en el fichero 'faces-config.xml'.

```
<application>
   <!– Renderiza las paginas jspx como Facelets-->
   <view-handler>com.sun.facelets.FaceletViewHandler</view-handler>
</application>
```

Los objetos dirección que aparecen en diferentes páginas de la aplicación son un ejemplo de esta funcionalidad.

- *Carpeta WebContent / Web-inf*

o faces-config.xml: es el archivo de configuración central de JavaServer faces. Aquí se define el flujo de trabajo de la aplicación (quien resuelve las peticiones), el manejo de las clases bean por JSF (en nuestro ejemplo, los beans son manejados por Spring, por lo que aquí sólo sirve para dar soporte a la autoayuda en el Idle de desarrollo) y otras cosas más como las reglas de navegación para los flujos de trabajo entre páginas.

```
…<faces-config>
<application>
        <!-- Dirige las peticiones de Beans a Spring–>
        <variable-resolver>org.springframework.web.jsf.PfcVariableResolver</variable-resolver>
        <!-- Renderiza las paginas jspx como Facelets-->
        <view-handler>com.sun.facelets.FaceletViewHandler</view-handler>
</application>
<!-- La parte de Declaración de Beans es útil para tener ayuda contextual en el momento de desarrollo –>
<!-- Faces no se encarga de recuperar Beans, lo hace Spring-->
<managed-bean>      <managed-bean-name>terceroEdit</managed-bean-name>
                    <managed-bean-class>com.pfc.admin.pages.TerceroEdit</managed-bean-class>
                    <managed-bean-scope>none</managed-bean-scope>
</managed-bean>
<managed-bean>
...
<!-- La parte destinada a las regla de navegación-->
<navigation-rule>
        <display-name>TercerosList</display-name>
        <navigation-case>    <from-outcome>TercerosList</from-outcome>
                             <to-view-id>/TercerosList.jspx</to-view-id>
        </navigation-case>
</navigation-rule>
<navigation-rule>
...
</faces-config>
```

o ApplicationContext.xml: En este fichero se define el contesto de funcionamiento de nuestra aplicación a nivel de Spring. En este sentido, los aspectos más interesantes Son

Especificación de los parámetros de conexión a la base de datos:

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
                <property name="driverClassName"><value>com.mysql.jdbc.Driver</value></property>
                <property name="url"><value>jdbc:mysql://127.0.0.1:3306/administracion</value></property>
                <property name="username"><value>root</value></property>
                <property name="password"><value>Uq3h4aiW</value></property>
</bean>
```

Para poder utilizar los mecanismos de persistencia de Hibernate se debe inicializar el entorno Hibernate y obtener un objeto session utilizando la clase SessionFactory de Hibernate. El siguiente fragmento de código ilustra la definición de la factoría de objetos de sesión:

```
<!-- Hibernate SessionFactory Definition -->
        <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
          <property name="entityInterceptor"><ref local="hibernateInterceptor"/></property>
          <property name="mappingResources">
            <list> <value>com/pfc/admin/entities/Albaran.hbm</value>
                   <value>com/pfc/admin/entities/Centro.hbm</value>
                   <value>com/pfc/admin/entities/Empresa.hbm</value>
                   <value>…</value>
            </list>
          </property>
          <property name="hibernateProperties">
            <props>
             <prop key="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</prop>
             <prop key="hibernate.show_sql">false</prop>
             <prop key="hibernate.cglib.use_reflection_optimizer">true</prop>
             <prop key="hibernate.cache.provider_class">org.hibernate.cache.HashtableCacheProvider</prop>
            </props>
          </property>
          <property name="dataSource"> <ref bean="dataSource"/>  </property>
        </bean>
<!-- Hibernate Template Defintion -->
        <bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
          <property name="sessionFactory"><ref bean="sessionFactory"/></property>
        </bean>
```

Define una entidad que intercepta los objetos recuperados por Hibernate. Esto permite, a través del AdminAccess poder inyectar todos los DAO. De esta forma están disponibles desde las entidades (función onLoad del objeto HibernateInterceptor). Esto permite poder pedir objetos a Spring desde las entidades de negocio, que aunque quizá no es muy purista, si facilita ciertas acciones dentro de estos objetos (pedir un inventario al almacenar un albarán).

```
  <bean id="hibernateInterceptor" class="com.pfc.admin.base.HibernateInterceptor">
          <property name="adminAccess"><ref local="adminAccess"/></property>
  </bean>
```

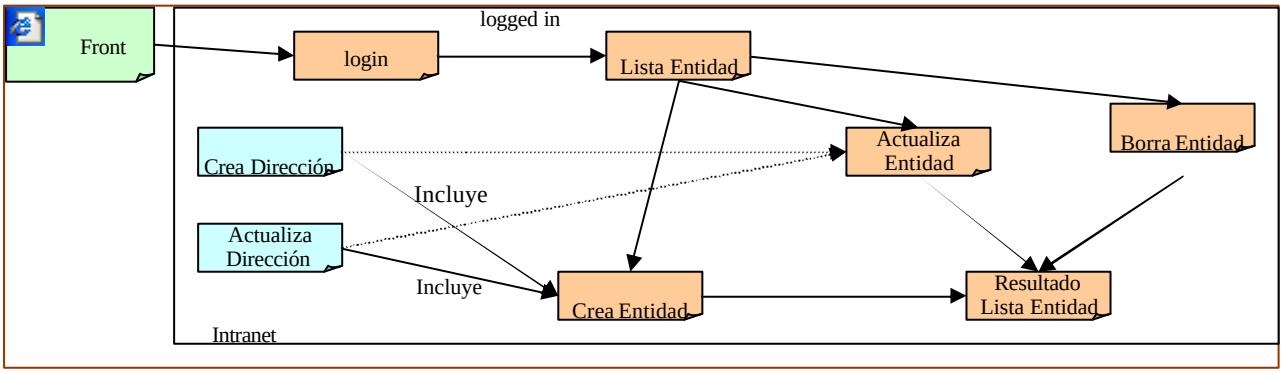
Un tema importante es la definición de cómo Spring tiene que 'confeccionar' un determinado objeto cuando este sea requerido por el sistemas. Esta configuración se realiza dentro de este fichero y se define a partir de tags 'property' dentro de la definición de cada bean. A continuación se muestra, como ejemplo, como construiria Spring un objeto albaran. La clase que define las reglas de negócio de este objeto las define el atributo 'class'. Por otro lado, se compone de otro objeto 'AdminAccess' (property name="adminAccess"), a continuación se indica que la definición de este objeto se encuentra en localmente dentro del bean AdminAccess (ref local="adminAccess"). Este razonamiento continua en cascada a partir de la definición sucesiva de los beans que compone el albaran. Es como si se tratase de una receta de cocina, donde necesitamos un conjunto de ingredientes para realizar el plato, y asu vez estos ingredientes pueden necesitar su propio proceso de elaboración, al juntarlo todo acabamos teniendo el plato final elaborado:

```
<!-- Albaran Definición -->
        <bean id= "albaran"  class="com.pfc.admin.entities.Albaran"  scope="prototype" >
                <property name="adminAccess"><ref local="adminAccess"/></property>
        </bean>
        <!-- AdminAccess Definición –>
        <bean id="adminAccess" class="com.pfc.admin.base.AdminAccess">
          <property name="logon"><ref local="logon"/></property>
          <property name="albaranDAO"><ref local="albaranDAO"/></property>
          ...<!-- ...resto de DAOS-->
        </bean>
        <!-- Albaran DAO Definición -->
        <bean id= "albaranDAO"  class="com.pfc.admin.entities.DAO.AlbaranDAO">
                <property name="hibernateTemplate"><ref local="hibernateTemplate"/></property>
        </bean>
        <!-- Hibernate Template Definición –>
        <bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
          <property name="sessionFactory"><ref bean="sessionFactory"/></property>
        </bean>
        <!-- Hibernate SessionFactory Definición –>
        <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
          ...<!– ...esta puesto en uno de los cuadros anteriores–>
        </bean>
```

.. Finalmente tenemos el fichero web.xml que contiene información de despliegue de la aplicación y se encuentra suficientemente comentado para comprender su funcionamiento

10.7 Navegación entre Páginas

La navegación entra páginas en bastante sencilla y se realiza a través del menú incorporado. A continuación se presenta un esquema básico de la navegación dentro de una de los mantenimientos que incorpora la aplicación. El resto de páginas son listadas por lo que su navegación es más sencilla.



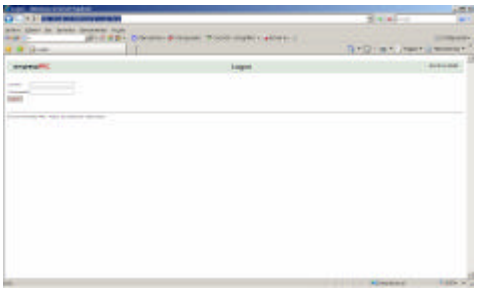
10.8 Pantallas

Todas las pantallas están diseñadas siguiendo un estilo uniforme que permita al usuario identificar de forma rápida la información presentada. Una visión de arriba hacia debajo de una página muestra:

- En la parte superior se muestra el logo de la empresa, el título de la página que se está visualizando y la fecha.
- A continuación se presenta el menú de opciones en forma de desplegable.
- Parte destinada a mensajes de la aplicación para retroalimentar las acciones de usuario.
- Parte destinada al filtrado de datos a mostrar. Sólo para páginas de tipo lista.
- Parte destinada a datos.
- Pie de la página con datos corporativos.

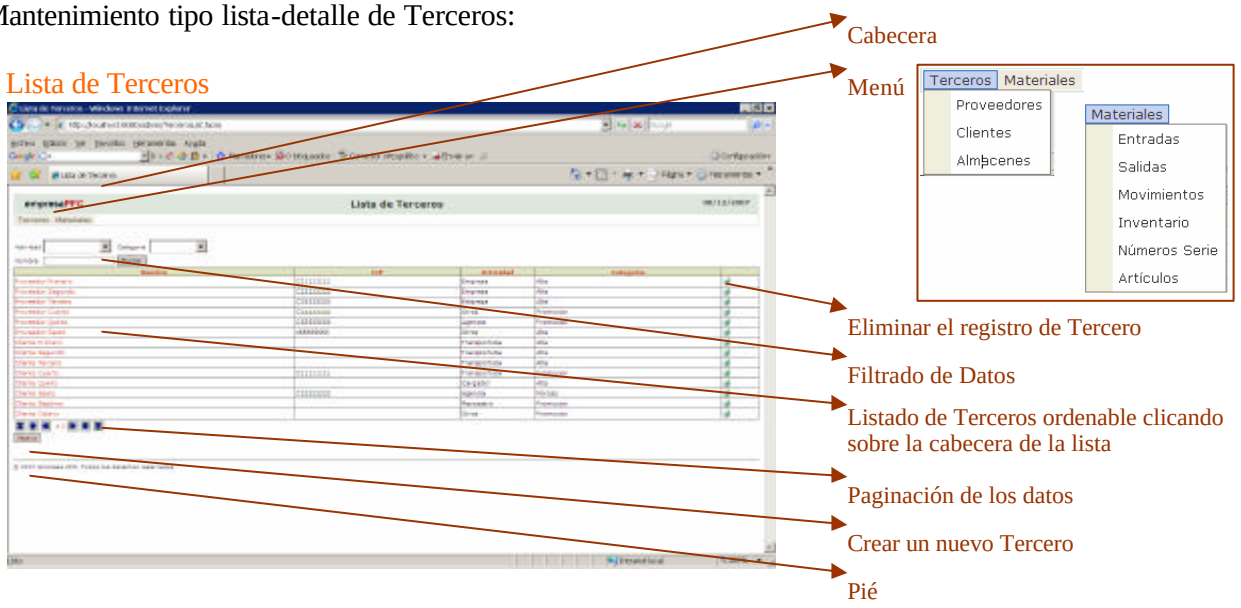
Si iniciamos la aplicación en un servidor local, podremos acceder a la página de inicio introduciendo el siguiente enlace en un navegador:

http://localhost:8080/admin/Login.faces Autentificación (pruebas, pruebas)



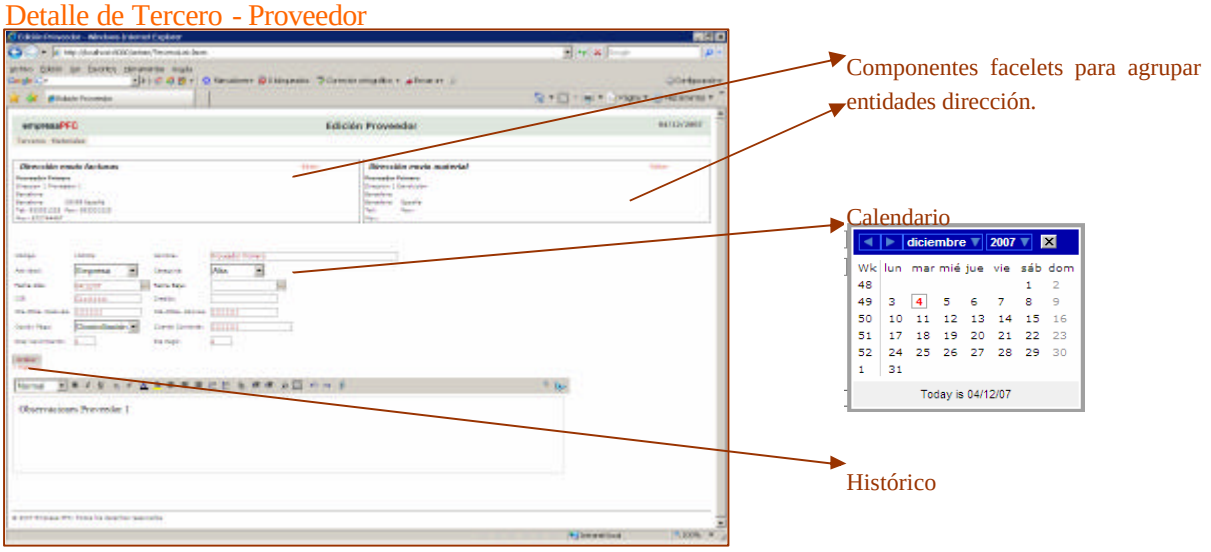
Una vez autentificados, el sistema nos redirige a la lista de Terceros (como ya hemos comentado como tercero se entiende tanto proveedores, clientes y almacenes). A través de menú podemos acceder a listas separadas de cada una de estas entidades. El comportamiento de este mantenimiento es muy parecido a como se realiza el de *Artículos*, que se encuentra bajo la opción de menú *Materiales*, por lo que únicamente se describirá uno de estos procesos

- Mantenimiento tipo lista-detalle de Terceros:



Tanto si clicamos sobre una línea de la columna nombre de la lista de Terceros, como si clicamos sobre el botón nuevo, el sistema de navegación nos redirige a la pantalla de detalle de Tercero. A continuación se muestra la pantalla de detalle de un Proveedor:

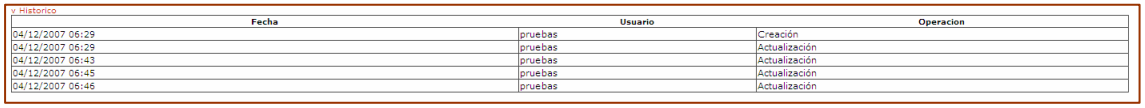
Detalle de Tercero - Proveedor



En está página aparecen componentes interesante. Justo después del menú se muestran componentes facelets para agrupar entidades dirección. Permite grabar datos en la entidad 'dirección dentro del mantenimiento de 'Terceros'. Permite reutilización. Se comportan de manera independiente, es como tener un mantenimiento dentro de otro, permitiendo creación, edición y borrado:

Como soporte a la entrada de fechas se ha incluido un objeto de faces de tipo calendario.

Otro aspecto interesante es la opción 'histórico'. Al clicar en esta opción aparece un listado que pretende resumir la vida de esta entidad en cuanto a su creación y posteriores modificaciones:

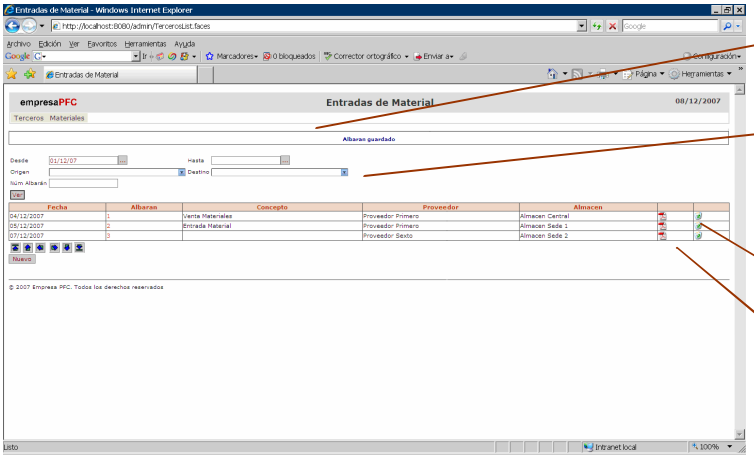


Histórico

Fecha	Usuario	Operacion
04/12/2007 06:29	pruebas	Creación
04/12/2007 06:29	pruebas	Actualización
04/12/2007 06:43	pruebas	Actualización
04/12/2007 06:45	pruebas	Actualización
04/12/2007 06:46	pruebas	Actualización

- Mantenimiento tipo lista-detalle de entrada de materiales:

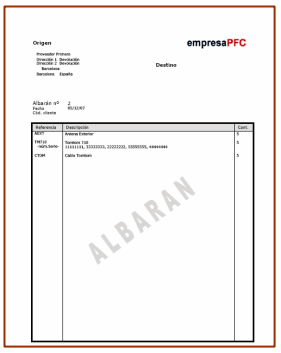
Lista de Entrada de Material



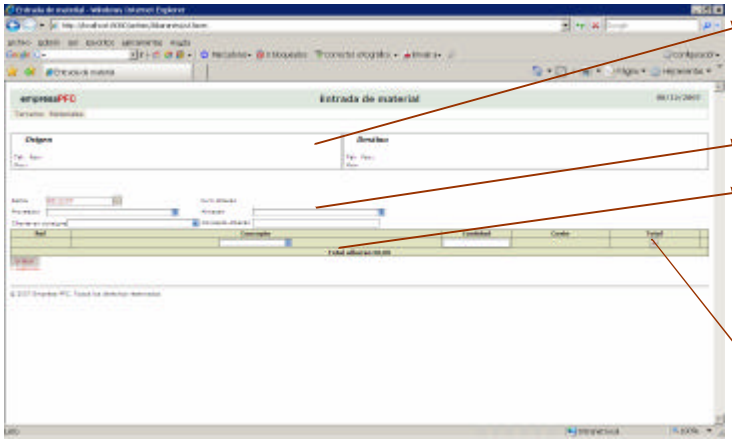
la pantalla pantalla está asociada a la entrada de materiales tanto desde un proveedor como desde un cliente. Una entrada de material genera un documento llamado albarán. Este documento es consultable en forma de PDF. A continuación se presentan las pantallas de lista y detalle de las entradas de material. En esta pantalla destacan los controles Ajax que permiten el filtrado de datos según '*Origen*' o '*Destino*'. Además se ha configurado de forma que a partir de un trozo de texto introdución, proponen el texto que lo contiene:

Propuestas a partir del texto insertado 'vee'

Otro punto interesante es la exportación del albarán a un documento PDF. Esto puede realizarse desde la lista de entradas de material. El documento resultante tendrá un aspecto:



Nueva de Entrada de material



Componentes facelets de direcciones de origen y destino. Se asignan en función del proveedor y el almacén

Datos genéricos de la entrada de material. Importante rellenarlos todos. Faltan algunas validaciones de campos vacíos.

Relación de artículos asociados al movimiento de entrada. En función del tipo de artículo varía el proceso. Para artículos asociados a un número de serie es obligatorio informarlos clicando sobre el campo cantidad, este campo se actualiza automáticamente en función de los números de serie entrados.

Para informar nueva entrada o entrar números de serie de la entrada actual, para ello hay, luego, hay que clicar en el campo cantidad

Es necesario indicar la lista de artículos que forman parte de la entrada de material. Los artículos pueden ser de dos tipo, a granel o con número de serie. Para los primeros es suficiente informar el artículo y la cantidad. Para los segundos, el sistema es un poco más complicado ya que hay que ir indicando los números de serie, la cantidad se incrementa automáticamente. Además se entiende que el usuario es experto y sabe que artículo son con número de serie y cuales no. Una mejora futura es indicar esta particularidad por pantalla. Pasos para entrar los artículos

Primero rellenar datos de cabecera

Inserción de un artículo a granel + cantidad y click sobre botón + (se admite borrado)

Inserción artículo de número de serie (sin cantidad) clicar +. Ahora es necesario introducir los números de serie uno a uno --> Clicar sobre la cantidad.

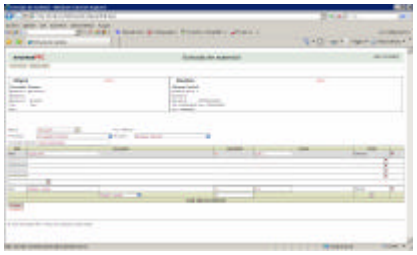
Ref	Concepto	Cantidad	Coste	Total	
AEXT	Antena Exterior	5	30.0	€150,00	✕
TM710	Tomtom 710	0	348.36	€0,00	✕
	Tomtom 710	5		+	
Total albaran €150,00					

Entrar los diferentes números de serie (p. ej: pistola lectora código de barras).Añadir nuevo numero de serie con el +. La cantidad se actualiza automáticamente. Se ha añadido además un nuevo artículo granel.

Ref	Concepto	Cantidad	Coste	Total	
AEXT	Antena Exterior	5	30.0	€150,00	✕
TM710	Tomtom 710	5	348.36	€1.741,80	✕
11111111					✕
22222222					✕
33333333					✕
44444444					✕
55555555					✕
	+				
CTOM	Cable Tomtom	5	20.0	€100,00	✕
	Cable Tomtom	5		+	
Total albaran €1.991,80					

Visualización de un movimiento de entrada de material. Permite cambiar dirección y se queda guardada de forma global. Además se pueden cambiar todos los valores:

Visualización de Entrada de material

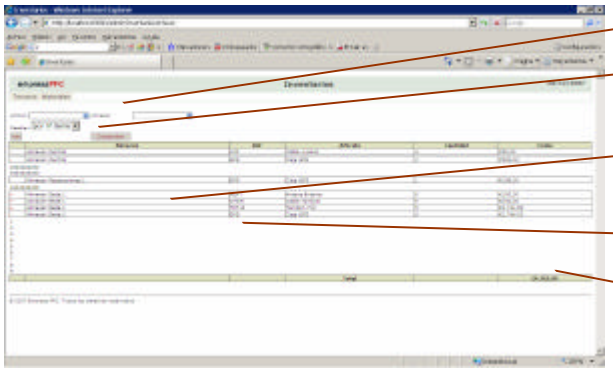


Las pantallas asociadas a las salidas de material y a los movimientos son muy parecidas a las de entrada y funcionan de forma muy similar, por ello no se incluyen.

- Pantalla Inventario:

Muestra el estado actual de existencias de artículos de los diferentes almacenes.

Visualización de Inventario



Criterios de Selección

Forzar validación del inventario: ver sicuadrán los datos de entradas y salidas

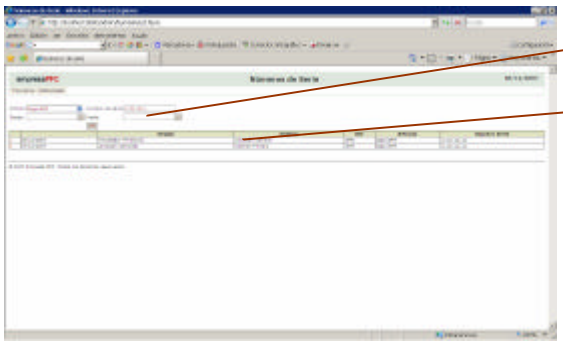
Detalle agrupado de existencias por artículo

Artículos con número de serie en el almacén seleccionado

Valor total de los artículos de los diferentes almacenes

Esta pantalla pretende ser un informe que sirva a los usuarios para tener una visión global del estado de existencias de artículos.

- Pantalla de números de Serie:



Criterios de Selección

Seguimiento de la evolución de un determinado artículo de tipo número de serie. Se muestra un listado con todos los albaranes implicados en función de los diferentes movimientos que ha podido sufrir el artículo seleccionado.

Sirve para seguir la pista de un artículo de tipo número de serie. Nos referimos a las posibles entradas y salidas que un determinado artículo ha podido sufrir. Por ejemplo, un artículo puede enviarse a un cliente, luego volver por estar estropeado o averiado, ir a reparar, volver de la reparación y retornar al cliente

11 CONCLUSIÓN

Actualmente estamos en un momento de estallido social en cuanto a tecnologías de la información se refiere: la brecha digital que poseía el país se está consiguiendo reducir gracias a la llegada de la banda ancha a los hogares y a las necesidades de la gente en cuanto a servicios tecnológicos ya sean profesionales o de ocio.

Con este proyecto se ha querido reflejar el gran abanico de posibilidades ante las que se enfrenta un profesional en el momento de inicial un proyecto de construcción de una aplicación Web en Java. Para ello se han repasado conceptos básicos de diseño como la arquitectura, los patrones o las especificaciones y se han estudiado algunas de las herramientas de soporte para el desarrollo de estos proyectos, los frameworks. Además se ha diseñado una solución tecnológica que satisface unas necesidades perfectamente existentes en el mundo real y que pretende ejemplificar un caso de uso de lo estudiado. Para ello se ha estudiado la manera de realizar un proyecto Web con una puesta en marcha rápida, integrando tecnologías de desarrollo punteras y fáciles de usar y sobretodo robusto y escalable.

Es interesante reflexionar sobre las dificultades encontradas en la redacción del presente documento. Se ha tenido que realizar un gran trabajo de investigación y sobretodo una gran labor de aprendizaje de tecnologías. La información y el número de herramientas existente es prácticamente inabarcable por lo que se ha tenido que realizar una importante tarea de filtrado. También existen carencias de información en determinados campos. Pero lo que más destacaría sería la visión global de la situación actual del sector, que se ha conseguido así como la organización de conceptos y tecnologías adquirida. A nivel personal es gratificante realizar un proyecto que sigue una corriente o tendencia socio/tecnológica. Gracias a esto he aprendido que el mundo de las nuevas tecnologías de Internet se mueve hacia un punto en el que para publicar no tienes porque tener conocimientos informáticos sino simplemente ganas de participar, y esto hace que se esté caminando hacía la utópica sociedad del conocimiento. Esta explosión en cuanto a publicación de contenidos y de conocimiento, llámese comunidades de código libre, foros, blogs, “spaces” o wikis hace que casi se pueda comparar la época en la que vivimos con otras grandes revoluciones tecnológicas por las que se ha pasado.

Como reflexión final, queda claro que las decisiones a tomar previas a la realización de un proyecto de este tipo, tanto tecnológicas como de diseño, no son una tarea nada fácil. Además son críticas para el resultado final. Es importante que el profesional esté al día de la evolución tecnológica del sector y que tenga una capacidad y experiencia adecuada para realizar satisfactoriamente este tipo de proyectos.

11.1 Líneas Abiertas

Las líneas abiertas con este proyecto son enormes. Existe la posibilidad de profundizar en cualquier tecnología de las citadas o en cualquier otra similar. A nivel de aplicación se puede profundizar en la aplicación presentada o realizar otras con otros diseños y tecnologías.

Además la constante evolución del sector va ha ir generando nuevas líneas de investigación que es importante seguir para mantenernos profesionalmente al día y poder atacar proyectos de manera solvente y competitiva aprovechando al máximo las posibilidades existentes.

12 BIBLIOGRAFÍA

Los enlaces más destacados consultados para realizar el presente documento son

http://java.sun.com
http://struts.apache.org/
http://tapestry.apache.org/
http://java.sun.com/javaee/javaserverfaces/
http://myfaces.apache.org/
http://wicket.apache.org/
http://www.springframework.org/docs/reference/mvc.html
http://turbine.apache.org/
http://jakarta.apache.org/velocity
http://maven.apache.org/
http://db.apache.org/torque/
http://cocoon.apache.org/
http://velocity.apache.org/
http://www.springframework.org/
http://java.sun.com/products/ejb/
http://hivemind.apache.org/
http://excalibur.apache.org/
http://www.jboss.com/products/seam
http://java.sun.com/products/jdbc/overview.html
http://www.hibernate.org/
http://ibatis.apache.org/
http://www.castor.org/
http://db.apache.org/torque/
http://www.acegisecurity.org/
http://www.hdiv.org/
http://www.junit.org/
http://java-source.net
http://www.java-samples.com
http://www.jcp.org
http://www.adictosaltrabajo.com/tutoriales
http://raibledesigns.com/rd/date/20070905
http://rollerjm.free.fr
http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html
http://www.programacion.com
http://www.jcp.org
http://www.opensymphony.com
http://www.google.es
http://appfuse.org
http://java-source.net
http://www.javahispano.org

Este documento contiene enlaces a esta web con la finalidad de aclarar determinados conceptos

http://es.wikipedia.org

Libros:

Core JSF, David Geary and Cay Horstmann
Spring Live, Matt Raible
Craig Larman.'Uml y Patrones'. ed 1999 Prentice Hall
Booch, Jacobson,Rumbaugh.'El Lenguaje Unificado de Modelado UML. Ed Wesley,Addison.

13 ANEXO 1

13.1 Instalación Entorno

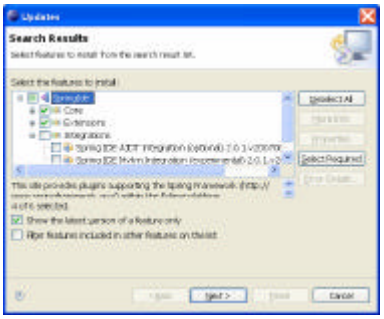
Los pasos para preparar el entorno de desarrollo para el proyecto son:

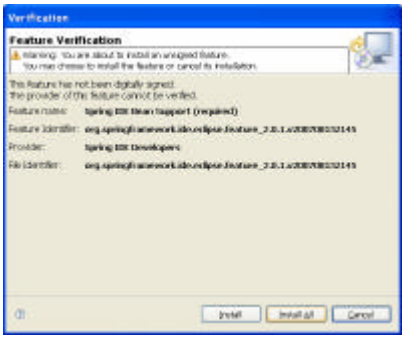
1. Descompactar el último WTP (p.e. wtp-all-in-one-sdk-R-1.5.4-win32.zip) en una carpeta propia del disco del PC.
2. Añadir el plugin de HibernateSynchronizer a la carpeta plugins (dentro del zip: plugins/com.hudson.hibernatesynchronizer_3.1.9).
3. Crear un acceso directo del fichero "eclipse.exe" en la carpeta del PC
4. Editar el fichero eclipse.ini y poner -Xmx512m en vez de –Xmx256m. Esto hace que no tenga problemas de memoria. Si esto no corrige los problemas, se puede modificar el acceso directo al programa con la llamada:

   'C:\eclipse_R201\eclipse\eclipse.exe -vmargs -Xms512m -Xmx512m -XX:PermSize=128m -XX:MaxPermSize=128m'

5. Ejecutar el acceso directo
6. Help>Software Updates>Find and Install. Search for new features. Web Tools Platform (WTP) Updates. Instalar JSF. Esta operaciónpuede tardar un tiempo considerable (hasta 20 – 30 min).Reiniciar.

7. Buscar actualizaciones de Faces ???
8. Help>Software Updates>Find and Install. Search for new features to install. New remote site, SpringIde con la url http://springide.org/updatesite/ Reiniciar





9. Instalar tomcat 5.5.
10. Instalar jdk-6u1-windows-i586-p.exe.
11. Crear un servidor Tomcat 5.5(Window>Preferentes>Server>Installed Runtimes).
12. Acceder al proyecto.

13. Motor de base de datos MySql Server 5.0