

## **PROYECTO FIN DE CARRERA**

DEFINICIÓN DE UN FRAMEWORK DE DESARROLLO DE APLICACIONES J2EE MEDIANTE EL DISEÑO E IMPLEMENTACIÓN DE UNA HERRAMIENTA DE GENERACIÓN AUTOMÁTICA DE CRUDs DE ENTIDADES ADAPTADAS A UN GESTOR DE PERSISTENCIA.

*DEFINITION OF A DEVELOPMENT FRAMEWORK FOR J2EE APPLICATIONS VIA DESIGN AND IMPLEMENTATION OF AN AUTOMATIC GENERATION CRUDS (CREATE, READ, UPDATE, DELETE) TOOL OF ENTITIES FROM A PERSISTENT MANAGER.*

AUTOR: CARLOS YAGÜE MARTÍN-LUNAS

DIRECTOR DEL PROYECTO: ANTONIO MAÑA GÓMEZ

# ÍNDICE

1. OBJETIVOS .....	4
1.1. Introducción a la arquitectura de CRUD-FW .....	5
1.2. Situación actual y necesidades que han motivado el desarrollo de este framework .....	7
1.3. Aportes y ventajas que cubre CRUD-FW .....	9
2. ANTECEDENTES .....	10
2.1. Model-1 .....	10
2.2. Model-2 .....	10
2.3. Desarrollo Rápido de aplicaciones (RAD) .....	10
2.4. Casos prácticos .....	11
3. TECNOLOGÍAS DE BASE.....	13
A continuación pasamos a describir el catálogo completo de tecnologías utilizadas en este proyecto: .....	13
3.1. Entorno de desarrollo.....	13
3.2. IDEs (Entorno de Desarrollo Integrado) .....	13
3.3. Apache Maven.....	14
3.4. Servidores de Aplicaciones J2EE.....	17
3.5. Servidores de Bases de Datos .....	17
3.6. Sistemas de persistencia .....	19
3.6.1. ORM: Mapeo objeto-relacional.....	19
3.6.2. El problema .....	19
3.6.3. Implementaciones.....	19
3.6.4. iBatis.....	20
3.6.5. Hibernate .....	23
3.6.6. JPA (Java Persistence API) .....	25
3.6.7. Hibernate vs. iBatis vs. JPA .....	29
3.7. Factoría de instancias y acople de distintas capas .....	30
3.7.1. Spring Framework .....	30
3.7.2. Spring en detalle .....	33
3.8. Frameworks MVC .....	34
3.8.1. Spring MVC .....	34
3.8.2. Apache Struts 2.....	37
3.8.3. JSF (JavaServer Faces).....	45
3.8.4 GWT (Google Web Toolkit) .....	53
3.8.5 Comparativa principales Frameworks MVC: Struts 2, JSF, y GWT .....	55
4. SOLUCIÓN APORTADA .....	57
4.1 Introducción.....	57
4.2 Concepto y definición.....	57
4.3 Generación automática de CRUDs a partir de entidades JPA.....	58
4.4 Arquitectura .....	58
4.5 Fases de desarrollo .....	61
5. MANUAL DE USUARIO .....	62
6 CONCLUSIONES.....	74
7 REFERENCIAS .....	76
7.1. RAD:.....	76
7.2. Eclipse: .....	76
7.3. Apache Maven:.....	76
7.4. Servidores J2EE: .....	77

7.5. SGBDs:.....	77
7.6. ORMs: .....	77
7.7. iBATIS: .....	77
7.8. Hibernate: .....	77
7.9. JPA: .....	78
7.10. Spring Framework: .....	78
7.11. CAS: .....	78
7.12. Spring Security: .....	78
7.13. Spring MVC: .....	78
7.14. Apache Struts 2: .....	78
7.15. JSF: .....	79
7.16. GWT: .....	79
7.17. Comparativas de Frameworks MVC: .....	79

## 1. OBJETIVOS

Este proyecto de fin de carrera pretende definir un *framework* (marco de trabajo) mediante una arquitectura que sirva de base para el desarrollo de aplicaciones J2EE (*Java 2 Enterprise Edition*) utilizando un proceso de desarrollo software tipo RAD (*Rapid Application Development*) sirviéndose para ello de herramientas de generación automática de código a medida que permitan disminuir el tiempo de desarrollo de dichas aplicaciones.

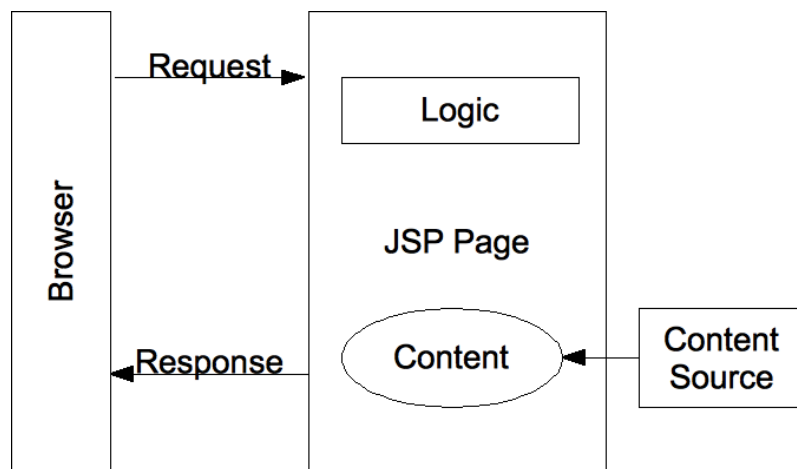
Las aplicaciones que se generarán de forma automatizada mediante la utilización de este *framework* serán de tipo CRUD (*Create, Read, Update, Delete*), es decir, serán aplicaciones preparadas para poder realizar operaciones de tipo *creación, lectura, actualización y borrado* de cada una de sus entidades del modelo. Por dicho motivo el framework desarrollado se ha denominado **CRUD-FW**.

Al tratarse de aplicaciones distribuidas por estar implementadas en J2EE, deberemos crear una arquitectura compatible con los componentes más estandarizados de la industria del Software, Como son Apache Tomcat, JBoss o Glassfish en lo que se refiere a servidores de aplicaciones y en cuanto al restricciones de almacenamiento, éstas serán cubiertas, con bases de datos como MySQL, Oracle y PostgreSQL.

Tradicionalmente este tipo de aplicaciones se han venido realizando mediante lo que se ha denominado **Modelo-1**:

El Modelo-1 se basa en la idea más simple en el desarrollo de aplicaciones Web. En él interviene únicamente un componente, un Servlet o JSP (Java Server Page) que es quien procesa la petición del usuario y se encarga de:

- Procesar la petición
- Validar los datos
- Aplicar la lógica de negocio asociada
- Generar la respuesta

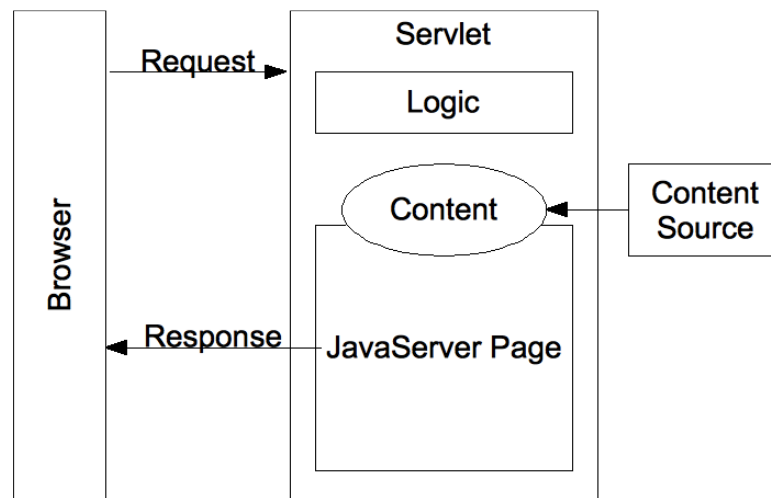


Este modelo sólo es válido para aplicaciones pequeñas, y es un modelo de desarrollo muy simple pero tiene grandes pegas:

- Poca reutilización de código. Muchas páginas pueden tener contenidos similares, así como procesos comunes, al no existir capas inferiores similares, difícilmente se va poder reutilizar o unificar bloques.
- Al mezclar lógica de negocio y vistas, se dificulta notablemente la adaptación a otros dispositivos de visualización como puede ser WML

Como evolución a este modelo, surge el **Modelo-2**:

Este modelo sí es recomendado para construir aplicaciones de cierta complejidad, e incluso para aplicaciones grandes. Se basa en la idea de separar la parte de presentación y la lógica de negocio. Este patrón de diseño es el precursor de uno de los patrones de desarrollo software más consolidados, el MVC (Model-View-Controller).

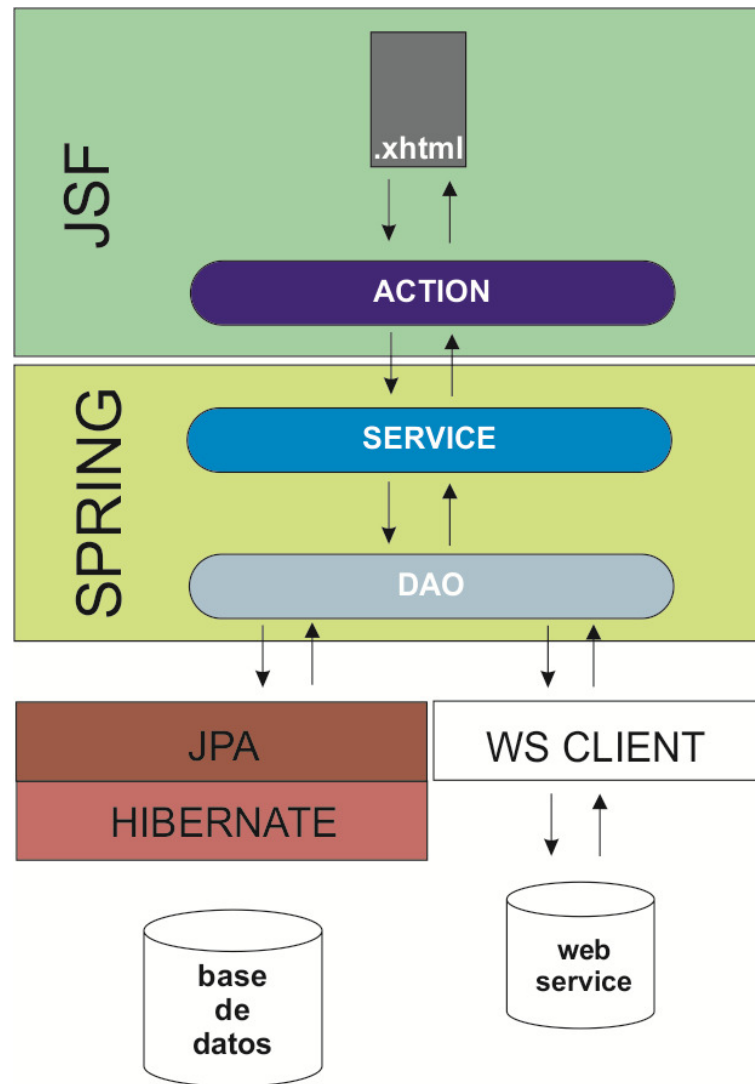


La idea es la misma, separar la parte de presentación (View) de la lógica de negocio (Model y Controller), mediante JSPs para las vistas y clases Java para procesar la lógica de negocio asociada.

Aparentemente estas formas de trabajar no son costosas, e incluso pueden ser la forma más rentable de construcción de aplicaciones simples. Pero los problemas aparecen cuando las necesidades de almacenamiento, lógica de negocio, incluso capas de vista y control se complican. Es en este punto donde aparecen muchos *frameworks* para disminuir costes de desarrollo, aportar mecanismos de reutilización de código y simplificando en capas la funcionalidad de la lógica de negocio. Con ello se consigue mejorar sustancialmente la calidad de las aplicaciones desarrolladas y disminuir costes en la fase de diseño arquitectural de dichas aplicaciones.

### ***1.1. Introducción a la arquitectura de CRUD-FW***

La arquitectura de este Framework estará distribuida en capas cumpliendo los patrones de diseño siguientes: MVC (Modelo-Vista-Controlador), DAO (Objeto de Acceso a Datos), todas ellas interconectadas con Spring Framework, que es un framework que facilita la inyección de dependencias de cada una de las capas. Más adelante se explicarán con más detalle cada una de estas capas.



Para el acceso a base de datos estas aplicaciones utilizarán JPA (*Java Persistence API*) lo que permitirá hacer una translación directa entre el modelo relacional (físico) y el modelo de objetos del aplicativo. Como ya veremos, la utilización de esta capa permitirá abstraer a nuestra aplicación del tipo de base de datos que se esté utilizando, conversión de distintos tipos del modelo físico, así como proporcionar un API de consultas muy potente.

Por encima de la capa de persistencia, se situarán las capas DAO y SERVICE que se encargarán del acceso a datos e implementación de lógica de negocio, respectivamente. Para declaración de las instancias de estas capas y su interconexión se utilizará Spring Framework, que como veremos más adelante, facilita la creación de instancias de tipo *JavaBean* (componentes compuestos de propiedades que pueden ser consultados y modificados mediante eventos) y sus inyecciones de dependencias.



Y por último, estaría el Framework MVC que utilizemos que en nuestro caso se tratará de JSF (JavaServer Faces) aunque también se podrían utilizar MVCs como Struts, OpenSymphony XWorks, Struts 2, Spring MVC, etc.

Llegado a este punto, como se puede observar se cuenta con un amplio abanico de tecnologías. Los principales aportes que ofrece un Framework de estas características son:

- Ofrecer al desarrollador un API de referencia que enmascare, en cierta medida, cada una de las peculiaridades de las posibles tecnologías que se pueden utilizar. Aportando como punto de partida una aplicación preparada para empezar a desarrollar y configurada, ofreciendo distintas posibilidades de elección de Frameworks MVC (Struts 2, JSF, etc), o bien un sistema de persistencia (iBatis, Hibernate, etc) concreto.
- Disminuir los tiempos de aprendizaje de los miembros del equipo de desarrollo al darle al desarrollador una estructura configurada y preparada para construir la aplicación que necesite. Uno de los grandes problemas de la producción de Software es la constante formación a la que deben estar sometidos los desarrolladores, debido a la continua aparición de nuevos Frameworks: Struts, XWork, Struts 2, JSF, etc.
- Una de las principales finalidades de CRUD-FW es la utilización de una herramienta de generación automática de código. Con dicha herramienta se consigue crear todo el código e inyecciones de dependencias de cada una de las capas de una entidad. Partiendo del *bean* del modelo mapeado en JPA, pasando por la generación de su DAO y Service específico hasta llegar a sus Actions con sus vistas XHTML concretas. Esta característica reduce al mínimo los tiempos de desarrollo, dejando pendiente al programador únicamente aquellas partes que constituyan parte del caso de uso específico de dicha entidad.

## 1.2. Situación actual y necesidades que han motivado el desarrollo de este framework

En la industria software actual, dentro de la rama de desarrollo J2EE, hay una amplia variedad de frameworks y tecnologías usadas en el desarrollo de software gestión. Hay frameworks MVC, para la capa de persistencia, para la capa de renderización de vistas y para gestionar la inyección de dependencias (Spring). En la siguiente tabla se muestra una relación de los frameworks más conocidos:

	Framework	Documentación oficial
MVC	Spring MVC	<a href="http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html">http://static.springsource.org/spring/docs/2.0.x/reference/mvc.html</a>

	Apache Struts	<a href="http://struts.apache.org/1.x/">http://struts.apache.org/1.x/</a>
	Opensymphony XWork	<a href="http://www.opensymphony.com/xwork/">http://www.opensymphony.com/xwork/</a>
	Apache Struts 2	<a href="http://struts.apache.org/2.2.1/index.html">http://struts.apache.org/2.2.1/index.html</a>
	JSF (JavaServer Faces)	<a href="http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html">http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html</a>
<b>Renderizado</b>	Apache Velocity	<a href="http://velocity.apache.org/engine/index.html">http://velocity.apache.org/engine/index.html</a>
	Opensymphony Sitemesh	<a href="http://www.opensymphony.com/sitemesh/">http://www.opensymphony.com/sitemesh/</a>
	Apache Tiles	<a href="http://tiles.apache.org/">http://tiles.apache.org/</a>
	Facelets	<a href="http://facelets.java.net/">http://facelets.java.net/</a>
<b>Persistencia</b>	Apache iBatis	<a href="http://ibatis.apache.org/">http://ibatis.apache.org/</a>
	Hibernate	<a href="http://www.hibernate.org/">http://www.hibernate.org/</a>
	JPA (Java Persistence API)	<a href="http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html">http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html</a>
<b>Inyecc. Dependencias</b>	Spring Framework	<a href="http://www.springsource.org/">http://www.springsource.org/</a>
	Google Guice	<a href="http://code.google.com/p/google-guice/">http://code.google.com/p/google-guice/</a>
<b>Etc</b>	Etc	

Como se puede observar, el número de frameworks disponibles es abrumador y no es viable que un desarrollador conozca todos estos frameworks. Es por ello que han surgido varias JSRs (Java Specification Requests) de Sun como respuesta a tanta diversidad, para tratar de unificar las interfaces de conexión entre las distintas capas. Ejemplos de estas especificaciones son JPA y JSF:

JPA (Java Persistence API) es una especificación que publicó Sun con la finalidad de que los sistemas gestores de persistencia tengan un comportamiento común, de tal forma que se pueda desacoplar con las capas superiores.

Con una filosofía similar, JSF (JavaServer Faces) constituye un marco de referencia para otra serie de frameworks, en este caso, MVC. JSF es una especificación y sobre esta se han realizado múltiples implementaciones como pueden ser:

- Apache MyFaces Tomahawk
- Apache MyFaces Sandbox
- Ajax4JSF
- IceFaces
- RichFaces
- PrimeFaces
- Etc

Pues bien, CRUD-FW es más que un Framework. Corresponde a una arquitectura que contempla en su estructura las especificaciones de Sun, JPA y JSF, así como Spring como mecanismo de gestión de inyecciones. Pero no sólo ofrece una arquitectura, también ofrece la posibilidad de facilitar el desarrollo de proyectos usando una herramienta de generación automática de código. Con esto se consigue minimizar los tiempos de desarrollo al ofrecer al programador un esqueleto funcional generando todas las capas de una entidad, desde su mapeo en el sistema gestor de persistencia, pasando por su DAO (Data Access Object), así como su MANAGER (servicio que ofrece la lógica de negocio necesaria), y su ACTION



(clase del modelo del MVC que es invocada desde la vista y devolverá el resultado esperado de cada caso de uso). Todo eso, sin necesidad de dominar todas las tecnologías que contempla la arquitectura del proyecto.

Además, las aplicaciones que se pueden construir con dicha herramienta, a corresponden a las aplicaciones más típicas en la industria del software de gestión, las tipo CRUD (*Create, Read, Update, Delete*), es decir, aquellas en las que existe, por cada entidad, una pantalla de listado y detalle. Ofreciendo las funcionalidades de creación de entidades nuevas (*Create*), consultar (*Read*), modificar (*Update*) y borrar (*Delete*) entidades existentes. Esta herramienta se corresponde una técnica de desarrollo RAD (*Rapid Application Development*) que se explicará más adelante.

Evidentemente, está funcionalidad tan básica no se corresponde con las mayoría de aplicaciones de gestión. Lo más probable es que en cada caso de uso se contemple alguna particularidad adicional. Pero en ese caso, el programador partirá de un esqueleto completo y programado y únicamente tendrá que realizar las modificaciones que necesite en función del caso de uso.

Por último, hay que indicar que paralelamente al desarrollo de este proyecto, se han desarrollado algunos frameworks similares basados en el mismo objetivo. Destaca Spring ROO como uno de los más conocidos. Más adelante se explicará con detalla este Framework de desarrollo.

### ***1.3. Aportes y ventajas que cubre CRUD-FW***

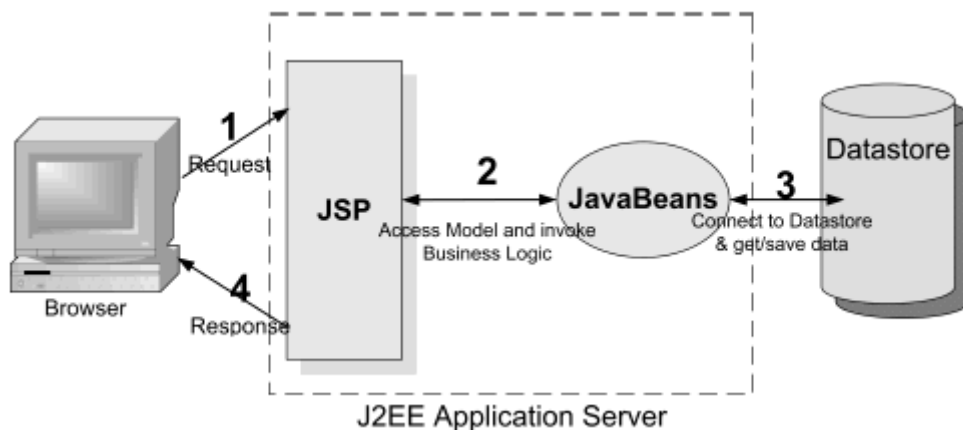
Resumiendo, los aportes que ofrece CRUD-FW son los siguientes:

- Constituir una arquitectura común a futuras aplicaciones J2EE de gestión usando las últimas tecnologías impuestas por Sun (JPA y JSF)
- La fase de definición de la arquitectura del proyecto se reduciría notablemente, suponiendo un ahorro de costes, al reutilizar componentes fiables, ya probados
- Ofrecer una herramienta de generación automática de código, disminuyendo costes y tiempos de desarrollo
- Ofrecer a un programador inexperto un marco de referencia otorgándole esqueletos funcionales completos de forma automática, y por consiguiente, reduciendo su tiempo adaptación a las tecnologías empleadas en dicha arquitectura.

## 2. ANTECEDENTES

### 2.1. Model-1

El Modelo-1 es la arquitectura más simple a la hora de desarrollar aplicaciones web basadas en JSP (Java Server Pages). No puede ser más fácil. En el modelo 1, el navegador accede directamente a las páginas JSP. En otras palabras, los usuarios las peticiones son manejadas directamente por el JSP. Vamos a ilustrar el funcionamiento de este modelo de arquitectura con un ejemplo. Considere la posibilidad de una página HTML con un hipervínculo a una página JSP. Cuando el usuario hace clic en el hipervínculo, la JSP se invoca directamente. El contenedor de servlets ejecuta el servlet Java resultante. La JSP contiene código incrustado y las etiquetas para acceder al modelo JavaBeans. El modelo de JavaBeans contiene atributos para la celebración de los parámetros de la petición HTTP desde la cadena de consulta. Además, contiene la lógica para conectar con el nivel medio o directamente a la base de datos usando JDBC para obtener los datos adicionales necesarios para mostrar la página. La JSP se representa como HTML usando los datos en el modelo de JavaBeans y otras clases de ayuda y las etiquetas.



### 2.2. Model-2

El Modelo 2 es una patrón de diseño más complejo usando en el desarrollo de aplicaciones web J2EE cuya principal característica es que separa claramente la visualización del contenido de la lógica de negocio usada para obtener y manipular el contenido. En ocasiones es confundido con el patrón MVC (model, view, controller) que veremos más adelante, al separar la presentación del procesamiento de una petición, pero este más minimalista que MVC. Este patrón es recomendado para aplicaciones de mediano y gran tamaño.

### 2.3. Desarrollo Rápido de aplicaciones (RAD)

Proceso de desarrollo de software que permite construir sistemas utilizables en poco tiempo, normalmente de 60 a 90 días, frecuentemente con algunas concesiones.

Principios en los que se basa:

- En ciertas situaciones, una solución utilizable al 80% puede producirse en el 20% de tiempo que se hubiera requerido para la solución completa.
- En ciertas situaciones, los requisitos de negocio de un sistema pueden satisfacerse aun cuando algunos de sus requisitos operacionales no se satisfagan.
- En ciertas situaciones, la aceptabilidad de un sistema puede determinarse en base a un conjunto mínimo de requisitos consensuados en lugar de la totalidad de los requisitos.

Problemas que soluciona:

- Con los métodos convencionales pasa un gran lapso de tiempo antes de que el cliente vea resultados
- Con los métodos convencionales el desarrollo llega a tardar tanto que para cuando el sistema está listo para utilizarse los procesos del cliente han cambiado radicalmente.
- Con los métodos convencionales no hay nada hasta que el 100% del proceso de desarrollo se ha realizado, entonces se entrega el 100% del software.

## ***2.4. Casos prácticos***

A continuación vamos a mostrar ejemplos prácticos de aplicaciones que han motivado la búsqueda de nuevos métodos de trabajo tanto en lo que se refiere a aspectos tecnológicos como metodológicos:

La utilización de modelos de programación no estructurados puede provocar situaciones inverosímiles, ejecución de consultas de base de datos desde las páginas JSP que pueden producir errores incontrolados y visibles por el usuario. Situaciones como esta pueden llegar a producir problemas serios de seguridad al dejar la puerta abierta a ataques por inyección de SQL, etc.

Por el contrario, contar con una arquitectura que separe en distintos niveles los mecanismos de acceso a datos, lógica de negocio y presentación tiene claros efectos de mejora de calidad del desarrollo de aplicaciones. Captura controlada de errores y excepciones, control de seguridad de acceso a determinadas funcionalidades, etc.

Otro punto a tener en cuenta es la elección de la metodología de desarrollo. Los métodos clásicos de construcción de aplicaciones suelen ser lentos, complejos y repetitivos. No se le puede enseñar al cliente el producto finalizado en el último momento puesto que puede que no tuviera claro en la fase de captura de requisitos que es lo que quería. Por ello es primordial que se planifique la presentación de productos parciales que le permitan visualizar el producto y darle la posibilidad de cambiar las especificaciones del producto final sin necesidad de que se finalice el desarrollo del producto.

Por otro lado, el historial de ejecución de proyectos debe servir para reducir los tiempos de proyectos futuros. Por ello, seguir un procedimiento de desarrollo que vaya conformando un catálogo de componentes reutilizables permitirá reducir los tiempos de desarrollo al disminuir la cantidad de código que se deba programar.

Un refinamiento importante al indicado en el párrafo anterior es la automatización del código de proyectos pasados, y en esa idea se basa CRUD-FW. Si se parte de una aplicación simple en la que para cada entidad se crea una página de listado y otra de detalle. Si por ejemplo, en un futuro se desarrolla una página de búsqueda y esta, pasa a formar parte del catálogo de páginas generadas por entidad de forma automática, poco a poco, se irán reduciendo los tiempos de desarrollo de los proyectos futuros.

### 3. TECNOLOGÍAS DE BASE

A continuación pasamos a describir el catálogo completo de tecnologías utilizadas en este proyecto:

#### 3.1. Entorno de desarrollo

Comenzamos con el entorno de desarrollo. Por entorno de desarrollo se entiende todo el conjunto de herramientas utilizadas tanto para el compilador, los servidores de pruebas, la herramientas de empaquetado, ejecución de test, etc

#### 3.2. IDEs (*Entorno de Desarrollo Integrado*)

Un **entorno de desarrollo integrado** (en inglés *integrated development environment*) es un programa informático por un conjunto de herramientas de programación.

Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, consiste en un editor de código, un compilador, un depurador y, posiblemente, un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. El lenguaje Visual Basic, por ejemplo, puede ser usado dentro de las aplicaciones de Microsoft Office, lo que hace posible escribir sentencias Visual Basic en forma de macros para Microsoft Word.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk u Objective-C.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, al que mediante plugins se le puede añadir soporte de lenguajes adicionales.

En la actualidad existen distintos entornos de desarrollo dentro de la comunidad J2EE. El más conocido y utilizado es Eclipse. Pero también existen otros como IDEA o Borland JBuilder.

En este proyecto se ha utilizado Eclipse Galileo durante su desarrollo y al tener como uno de los artefactos resultantes de este proyecto, una herramienta de generación automática de código, se explicará como configurar dicha herramienta sobre este entorno para ser incluida en el *workflow* de los equipos de desarrollo.

Eclipse es un entorno de desarrollo integrado de código libre multiplataforma al ser una aplicación que corre sobre la máquina virtual de java (JVM). Aunque Eclipse está reconocido

como el IDE J2EE por excelencia, también puede servir para otros lenguajes de programación como C++, ya que su mecanismo de plugins lo convierte en un IDE extensible. Este es el caso del plugin **CDT Project** que ofrece soporte para C y C++.

Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado **Eclipse Modeling Project**, cubriendo casi todas las áreas de **Model Driven Engineering**.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para **VisualAge**. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La base para Eclipse es la Plataforma de cliente enriquecido (del Inglés Rich Client Platform RCP).

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés *plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente a permitirle a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, permite a Eclipse trabajar con lenguajes para procesamiento de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos (SGBD). La arquitectura plugin permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente para soportar otros lenguajes de programación.

El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código. Mediante diversos plugins estas herramientas están también disponibles para otros lenguajes como C/C++ (Eclipse CDT) y en la medida de lo posible para lenguajes de script no tipados como PHP o Javascript. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de metadata en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

### **3.3. Apache Maven**

Apache Maven es una herramienta capaz de gestionar del ciclo de vida de un proyecto de Software (*Project Management Tool*). Es usada tanto por desarrolladores como por ingenieros de pruebas, arquitectos software y jefes de proyecto. En este proyecto constituye un pilar importante ya que la herramienta de generación automática que incorpora corresponde a un plugin Maven.

Apache Maven se puede confundir con Apache Ant al ofrecer la posibilidad de preprocesar los recursos, compilar, empaquetar, probar y distribuir a partir del código fuente un proyecto.

No obstante, Maven ofrece más características como mecanismos de generación de informes, publicación en un servidor Web, etc. Esto mejora la comunicación en un equipo de trabajo.

Formalmente y a un nivel más bajo de especificación, Maven es una herramienta de gestión de proyectos que cumplen con la especificación de **Project Object Model (POM)** que corresponde a un conjunto de estándares:

Un ciclo de vida del proyecto (*Project lifecycle*)

Un sistema de gestión de dependencias (*Dependency Management System*)

Cuando se define un proyecto Maven, hay que definirlo en base a la estructura de un fichero **pom.xml** (Project Object Model).

Maven está construido usando una arquitectura basada en plugins que permite que utilice cualquier aplicación controlable a través de la entrada estándar. En teoría, esto podría permitir a cualquiera escribir plugins para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, etcétera, para cualquier otro lenguaje. En realidad, el soporte y uso de lenguajes distintos de Java es mínimo. Actualmente existe un plugin para .Net Framework y es mantenido, y un plugin nativo para C/C++ fue alguna vez mantenido por Maven versión 1. Apache Maven ofrece un API de definición de **plugins** que pueden ser declarados dentro de la estructura del fichero POM.

Hay un amplio abanico de plugins Maven disponibles, entre los que destacan en la actualidad son:

Para ver la lista de plugins más actualizada, visualizar el repositorio Maven oficial en el subdirectorio [org/apache/maven/plugins](http://org.apache.maven/plugins).

Existe un conjunto de plugins básicos que son ejecutados durante cada una de las fases del ciclo de vida de construcción:

<a href="#">clean</a>	Elimina los compilados
<a href="#">compiler</a>	Compila código Java
<a href="#">deploy</a>	Despliega el artefacto en un repositorio remoto
<a href="#">failsafe</a>	Ejecuta los tests de integración JUnit
<a href="#">install</a>	Instala el artefacto en el repositorio local
<a href="#">resources</a>	Copia los recursos al directorio de salida
<a href="#">site</a>	Generate a site for the current project.
<a href="#">surefire</a>	Ejecuta los tests unitarios JUnit
<a href="#">verifier</a>	Verificación de ciertas condiciones de existencia de archivos (Utilidad de los tests de integración)

Maven además cuenta con una serie de plugins que son usados en función del tipo de empaquetado que se configure en el POM en la marca:

```
<project>
  <packaging> ${empaquetado.de.salida} </packaging>
</project>
```

Donde **\${empaquetado.de.salida}** puede ser: ear, ejb, jar, rar, war, app-client, shade.

Cada tipo de empaquetado tiene asociado un plugin que Maven ejecutará durante su fase **package**.

Hay también una serie de plugins que sirven para **generar informes** que al configurarlos en el POM se ejecutan en la fase `site` del ciclo de vida:

Plugin	Descripción
<a href="#">changelog</a>	Genera una lista de cambios recientes en el repositorio del SCM (Gestión de Configuración de Software)
<a href="#">checkstyle</a>	Genera un informe de Checkstyle: <a href="http://checkstyle.sourceforge.net/">http://checkstyle.sourceforge.net/</a>
<a href="#">javadoc</a>	Genera un informe Html con todo el Javadoc del proyecto
<a href="#">pmd</a>	Genera un informe de PMD: <a href="http://pmd.sourceforge.net/">http://pmd.sourceforge.net/</a>
<a href="#">surefire-report</a>	Genera un informe con los resultados de ejecutar los tests JUnit

Saliendo del ciclo de vida por defecto, hay muchas **herramientas** disponibles por defecto:

Plugin	Descripción
<a href="#">ant</a>	Genera un fichero de construcción Ant para el proyecto
<a href="#">antrun</a>	Ejecuta un conjunto de tareas Ant en una fase del ciclo de construcción Maven
<a href="#">archetype</a>	Genera un esqueleto con la estructura de un proyecto a partir de un arquetipo.
<a href="#">dependency</a>	Manipulación de dependencias: <code>copy</code> , <code>unpack</code> , <code>tree</code> , <code>analysis</code> , etc.
<a href="#">help</a>	Obtiene información acerca del entorno de trabajo del proyecto
<a href="#">patch</a>	Usa la herramienta Patch de GNU para aplicar ficheros patch al código fuente
<a href="#">release</a>	Genera una versión del proyecto actual, actualizando el POM y etiquetándola en el SCM
<a href="#">remote-resources</a>	Copia recursos remotos al fichero de salida
<a href="#">source</a>	Genera un JAR con los Fuentes del proyecto

Por otro lado, Maven precisa debe convivir con los **Entornos de Desarrollo Integrados (IDEs)**, por dicho motivo existen plugins que nos generan proyectos a partir de su definición en Maven a través de su correspondiente POM:

Plugin	Descripción
<a href="#">eclipse</a>	Genera un proyecto Eclipse a partir de la definición del POM del proyecto actual: los ficheros <code>.project</code> y <code>.classpath</code> y el directorio <code>.settings</code>
<a href="#">idea</a>	Crea y actualiza un espacio de trabajo en IDEA para el proyecto actual



Existen múltiples plugins desarrollados por organizaciones y empresas fuera de la comunidad Apache Maven. Destacan plugins como:

Plugin	Organismo responsable	Descripción
<a href="#">native</a>	Codehaus	Compila fuentes C y C++ con sus compiladores nativos.
<a href="#">sql</a>	Codehaus	Ejecuta scripts SQL
<a href="#">taglist</a>	Codehaus	Genera una lista de tareas basadas en etiquetas del código del tipo <code>TODO</code> o <code>FIXME</code> .
<a href="#">cargo</a>	<a href="#">Cargo Project</a>	Configura el despliegue sobre contenedores de aplicaciones J2EE
<a href="#">clover</a>	<a href="#">Atlassian Clover</a>	Genera informes Clover.
<a href="#">jaxme</a>	<a href="#">Apache Web Services Project</a>	Usa la implementación JaxMe de JAXB para generar fuentes Java a partir de esquemas XML.
<a href="#">jalopy</a>	<a href="#">Triemax</a>	Formateador de código fuente

### 3.4. Servidores de Aplicaciones J2EE

Como servidor de aplicaciones utilizaremos Apache Tomcat versión 5.5

Apache Tomcat funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation e implementa las especificaciones de Javax Servlets 2.4 y de JavaServer Pages (JSP) 2.0 de Sun Microsystems.

Tomcat es un proyecto de Software libre desarrollado y mantenido por la Apache Software Foundation . Tomcat es un servidor web con soporte de servlets y JSPs., pero no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

La jerarquía de directorios de instalación de Tomcat incluye:

- **bin** - arranque, cierre, y otros scripts y ejecutables
- **common** - clases comunes que pueden utilizar Catalina y las aplicaciones web
- **conf** - ficheros XML y los correspondientes DTD para la configuración de Tomcat
- **logs** - logs de Catalina y de las aplicaciones
- **server** - clases utilizadas solamente por Catalina
- **shared** - clases compartidas por todas las aplicaciones web
- **webapps** - directorio que contiene las aplicaciones web
- **work** - almacenamiento temporal de ficheros y directorios

### 3.5. Servidores de Bases de Datos

Como servidor de base de datos utilizaremos MySQL versión 5.0.91

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones. MySQL AB, desde enero de 2008 una subsidiaria de Sun Microsystems y ésta a su vez de Oracle Corporation desde abril de 2009, desarrolla MySQL como software libre en un esquema de licenciamiento dual.

El nombre de MySQL procede de la combinación de My, hija del cofundador Michael "Monty" Widenius, con el acrónimo SQL (según la documentación de la última versión en inglés). Por otra parte, el directorio base y muchas de las bibliotecas usadas por los desarrolladores tenían el prefijo My.

El nombre del delfín de MySQL es Sakila y fue seleccionado por los fundadores de MySQL AB en el concurso "Name the Dolphin". Este nombre fue enviado por Ambrose Twebaze, un desarrollador de software de código abierto africano, derivado del idioma SiSwate, el idioma local de Swazilandia y corresponde al nombre de una ciudad en Arusha, Tanzania, cerca de Uganda la ciudad origen de Ambrose.

Existen varias APIs que permiten, a aplicaciones escritas en diversos lenguajes de programación, acceder a las bases de datos MySQL, incluyendo C, C++, C#, Pascal, Delphi (via dbExpress), Eiffel, Smalltalk, Java (con una implementación nativa del driver de Java), Lisp, Perl, PHP, Python, Ruby, Gambas, REALbasic (Mac y Linux), (x)Harbour (Eagle1), FreeBASIC, y Tcl; cada uno de estos utiliza una API específica. También existe una interfaz ODBC, llamado MyODBC que permite a cualquier lenguaje de programación que soporte ODBC comunicarse con las bases de datos MySQL. También se puede acceder desde el sistema SAP, lenguaje ABAP.

MySQL es muy utilizado en aplicaciones web, como Drupal o phpBB, en plataformas (Linux/Windows-Apache-MySQL-PHP/Perl/Python), y por herramientas de seguimiento de errores como Bugzilla. Su popularidad como aplicación web está muy ligada a PHP, que a menudo aparece en combinación con MySQL. MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones. Sea cual sea el entorno en el que va a utilizar MySQL, es importante adelantar monitoreos sobre el desempeño para detectar y corregir errores tanto de SQL como de programación.

### 3.6. *Sistemas de persistencia*

#### 3.6.1. ORM: Mapeo objeto-relacional

El **mapeo objeto-relacional** (más conocido por su nombre en inglés, Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos, aunque algunos programadores prefieren crear sus propias herramientas ORM.

#### 3.6.2. El problema

En la programación orientada a objetos, las tareas de gestión de datos son implementadas generalmente por la manipulación de objetos, los cuales son casi siempre valores no escalares. Para ilustrarlo, considere el ejemplo de una entrada en una libreta de direcciones, que representa a una sola persona con cero o más números telefónicos y cero o más direcciones. En una implementación orientada a objetos, esto puede ser modelado por un "objeto persona" con "campos" que almacenan los datos de dicha entrada: el nombre de la persona, una lista de números telefónicos y una lista de direcciones. La lista de números telefónicos estaría compuesta por "objetos de números telefónicos" y así sucesivamente. La entrada de la libreta de direcciones es tratada como un valor único por el lenguaje de programación (puede ser referenciada por una sola variable, por ejemplo). Se pueden asociar varios métodos al objeto, como uno que devuelva el número telefónico preferido, la dirección de su casa, etc. Sin embargo, muchos productos populares de base de datos, como los Sistemas de Gestión de Bases de Datos SQL, solamente pueden almacenar y manipular valores escalares como enteros y cadenas, organizados en tablas normalizadas. El programador debe convertir los valores de los objetos en grupos de valores simples para almacenarlos en la base de datos (y volverlos a convertir luego de recuperarlos de la base de datos), o usar sólo valores escalares simples en el programa. El mapeo objeto-relacional es utilizado para implementar la primera aproximación.

El núcleo del problema reside en traducir estos objetos a formas que puedan ser almacenadas en la base de datos para recuperarlas fácilmente, mientras se preservan las propiedades de los objetos y sus relaciones; estos objetos se dice entonces que son persistentes.

#### 3.6.3. Implementaciones

Los tipos de bases de datos usados mayormente son las bases de datos SQL, cuya aparición precedió al crecimiento de la programación orientada a objetos en los 1990s. Las bases de datos SQL usan una serie de tablas para organizar datos. Los datos en distintas tablas están asociados a través del uso de restricciones declarativas en lugar de punteros o enlaces explícitos. Los mismos datos que pueden almacenarse en un solo objeto podrían requerir ser almacenados a través de varias tablas.

Una implementación del mapeo relacional de objetos podría necesitar elegir de manera sistemática y predictiva qué tablas usar y generar las sentencias SQL necesarias.

Muchos paquetes han sido desarrollados para reducir el tedioso proceso de desarrollo de sistemas de mapeo relacional de objetos proveyendo bibliotecas de clases que son capaces de realizar mapeos automáticamente. Dada una lista de tablas en la base de datos, y objetos en el programa, ellos pueden automáticamente mapear solicitudes de un sentido a otro. Preguntar a un objeto persona por sus números telefónicos resultará en la creación y envío de la consulta apropiada, y los resultados son traducidos directamente en objetos de números telefónicos dentro del programa.

Desde el punto de vista de un programador, el sistema debe lucir como un almacén de objetos persistentes. Uno puede crear objetos y trabajar normalmente con ellos, los cambios que sufran terminarán siendo reflejados en la base de datos.

Sin embargo, en la práctica no es tan simple. Todos los sistemas ORM tienden a hacerse visibles en varias formas, reduciendo en cierto grado la capacidad de ignorar la base de datos. Peor aún, la capa de traducción puede ser lenta e ineficiente (comparada en términos de las sentencias SQL que escribe), provocando que el programa sea más lento y utilice más memoria que el código "escrito a mano".

Un buen número de sistemas de mapeo objeto-relacional se han desarrollado a lo largo de los años, pero su efectividad en el mercado ha sido diversa. NeXT's Enterprise Objects Framework (EOF) fue una de las primeras implementaciones, pero no tuvo éxito debido a que estaba estrechamente ligado a todo el kit de NeXT's, OpenStep. Fue integrado más tarde en NeXT's WebObjects, el primer servidor web de aplicaciones orientado a objetos. Desde que Apple compró NeXT's en 1997, EOF proveyó la tecnología detrás de los sitios web de comercio electrónico de Apple: los servicios .Mac y la tienda de música iTunes. Apple provee EOF en dos implementaciones: la implementación en Objective-C que viene con Apple Developers Tools y la implementación Pure Java que viene en WebObjects 5.2. Inspirado por EOF es el open source Apache Cayenne. Cayenne tiene metas similares a las de EOF e intenta estar acorde a los estándares JPA.

Una aproximación alternativa ha sido tomada por tecnologías como RDF y SPARQL, y el concepto de "triplestore". RDF es una serialización del concepto objeto-sujeto-predicado, RDF/XML es una representación en XML de aquello, SPARQL es un lenguaje de consulta similar al SQL, y un "triplestore" es una descripción general de una base de datos que trabaja con un tercer componente.

Más recientemente, un sistema similar ha comenzado a evolucionar en el mundo Java, conocido como Java Persistence API (JPA). A diferencia de EOF, JPA es un estándar, y muchas implementaciones están disponibles por parte de distintos distribuidores de software. La especificación 3.0 de Enterprise Java Beans (EJB) también cubre la misma área. Otro ejemplo a mencionar es Hibernate, el framework de mapeo objeto-relacional más usado en Java que inspiró la especificación EJB 3.

En el framework de desarrollo web Ruby on Rails, el mapeo objeto-relacional juega un rol preponderante y es manejado por la herramienta ActiveRecord. Un rol similar es el que tiene el módulo DBIx::Class para el framework basado en Perl Catalyst, aunque otras elecciones también son posibles.

#### 3.6.4. iBatis

iBatis es un framework de persistencia para Java con bastantes diferencias respecto a **Hibernate**, el gran dominador en este segmento. También existen versiones para .NET y Ruby, pero nos centraremos únicamente en la versión Java.

La principal diferencia entre iBatis e Hibernate es que iBatis no es un ORM (Mapeo Objeto Relacional). Por lo tanto Hibernate genera el SQL para mapear objetos a tablas de la base de datos, mientras que en iBatis el SQL lo tendremos que escribir nosotros. Si tenemos un modelo de datos que dista mucho del modelo de negocio (nuestras clases Java) Hibernate nos puede dar bastantes dolores de cabeza.

Lo que sí hace iBatis es lo siguiente:

Ejecuta el SQL escrito por nosotros mediante JDBC (Java DataBase Connectivity), por lo que nos olvidamos de los múltiples try/catch.

Mapea propiedades de objetos a parámetros para las PreparedStatement (sentencias SQL parametrizables).

Mapea los resultados de una consulta a un objeto o una lista de objetos.

Veamos un ejemplo de uso:

Supongamos que contamos con una tabla Persona.

El *script* SQL que la generaría, **persona.sql**:

```
CREATE TABLE PERSONA (
PER_ID                NUMBER (5, 0) NOT NULL,
PER_NOMBRE            VARCHAR (40) NOT NULL,
PER_APELLIDO          VARCHAR (40) NOT NULL,
PER_FECHA_NACIMIENTO  DATETIME,
PER_PESO              NUMBER (4, 2) NOT NULL,
PER_ALTURA           NUMBER (4, 2) NOT NULL,
PRIMARY KEY (PER_ID))
```

Su *bean* asociado, **Persona.java**:

```
package org.crudfw.ejemplos.ibatis.dto;
//imports omitidos...
public class Persona {
private int id;
private String nombre;
private String apellido;
private Date fechaNacimiento;
private double peso;
private double altura;

public int getId () {
return id;
}
public void setId (int id) {
this.id = id;
}
//...asumimos que aquí tendríamos otros métodos get y set...
}
```

Y la forma en que iBatis realizaría la correspondencia entre ambos sería mediante este fichero de configuración:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMapConfig
PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
```

```

"http://ibatis.apache.org/dtd/sql-map-config-2.dtd">
<sqlMapConfig>
<properties
resource="ejemplo/sqlmap/maps/SqlMapConfigEjemplo.properties" />
<settings cacheModelsEnabled="true"
enhancementEnabled="true"
lazyLoadingEnabled="true"
maxRequests="32"
maxSessions="10"
maxTransactions="5"
useStatementNamespaces="false" />

<typeAlias alias="order" type="testdomain.Order"/>
<transactionManager type="JDBC" >
<dataSource type="SIMPLE">
<property name="JDBC.Driver" value="${driver}"/>
<property name="JDBC.ConnectionURL" value="${url}"/>
<property name="JDBC.Username" value="${username}"/>
<property name="JDBC.Password" value="${password}"/>
</dataSource>
</transactionManager>
<sqlMap resource="ejemplo/sqlmap/maps/Persona.xml" />
</sqlMapConfig>

```

Donde ejemplo/sqlmap/maps/Persona.xml sería:

```

<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap
PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
"http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap namespace="Persona">
<select id="getPersona"
resultClass="org.crudfw.ejemplos.ibatis.dto.Persona">
SELECT
PER_ID as id,
PER_NOMBRE as nombre,
PER_APELLIDO as apellido,
PER_F_NACIMIENTO as fechaNacimiento,
PER_PESO as peso,
PER_ALTURA as altura
FROM PERSONA
WHERE PER_ID = #value#
</select>
</sqlMap>

```

En resumen:

Tendremos un fichero de configuración de iBatis (*SqlMapConfig.xml* habitualmente) en el que indicaremos algunos parámetros de iBatis (como si está activo el cacheado), el **DataSource** (soporta varios tipos de **DataSource**) y los distintos mapeos SQL. Tendremos al menos un fichero de mapeo xml, en el que se indican las sentencias SQL, los parámetros a mapear o los resultados a mapear.

Tendremos una clase que implementa `SQLMapClient` que es el interface que contiene los métodos para realizar las sentencias SQL (`queryForObject`, `queryForList` o `delete` entre otros muchos). Esta clase la habremos obtenido mediante `SqlMapClientBuilder`.

Otro punto fuerte de iBatis es su integración con Spring. Para trabajar con iBatis integrado en Spring los pasos a seguir serían los siguientes:

El `DataSource` lo definiremos como un Bean en los archivos de definición de Spring. El resto del `SqlMapConfig.xml` y los xml de mapeo seguirían igual.

Spring proporciona el `SqlMapClientFactoryBean` y `SqlMapClientTemplate`. A la factoría le proporcionaremos el `DataSource` y al template le proporcionamos la factoría. Ese template es la clase que nos proporciona los mismos métodos que el `SQLMapClient` visto anteriormente. Por otro lado, existe una herramienta de generación automática de código para iBatis: **ibator**. Es un plugin para Eclipse que a partir de una tabla de BBDD es capaz de generar el XML de mapeo y las clases Java que representan al modelo de datos e incluso los DAO (Objetos de Acceso a Datos).

### 3.6.5. Hibernate

**Hibernate** es una herramienta de [Mapeo objeto-relacional](#) (ORM) para la plataforma Java y disponible también para [.Net](#) con el nombre de [NHibernate](#) que facilita el mapeo de atributos entre una [base de datos](#) relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos ([XML](#)) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL.

#### Características

Como todas las herramientas de su tipo, Hibernate busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la POO. Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por SQL. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución.

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

Hibernate ofrece también un lenguaje de consulta de datos llamado **HQL** (*Hibernate Query Language*), al mismo tiempo que una API para construir las consultas programáticamente (conocida como "*criteria*").

Hibernate para Java puede ser utilizado en aplicaciones Java independientes o en aplicaciones Java EE, mediante el componente **Hibernate Annotations** que implementa el estándar JPA, que es parte de esta plataforma.

Veamos un ejemplo. Sea `Evento.java` el bean de una tabla `EVENTO`:

```
package org.crudfw.ejemplos.hibernate.dto;

import java.util.Date;

public class Evento {
    private Long id;
    private String titulo;
    private Date fecha;

    public Evento() {}

    public Long getId() {
        return id;
    }
    private void setId(Long id) {
        this.id = id;
    }
    public Date getFecha() {
        return fecha;
    }
    public void setFecha(Date fecha) {
        this.fecha = fecha;
    }
    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
}
```

Y sea el `/org/crudfw/ejemplo/hbms/Evento.hbm.xml` el mapeo de dicha tabla:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="org.crudfw.ejemplos.hibernate.dto">
    <class name="Evento" table="EVENTOS">
        <id name="id" column="EVENT_ID">
            <generator class="native"/>
        </id>
        <property name="fecha" type="timestamp">
```



```
        column="EVENT_FECHA"/>
    <property name="titulo"/>
</class>
</hibernate-mapping>
```

### 3.6.6. JPA (Java Persistence API)

JPA (Java Persistence API) es el API de persistencia desarrollado por Sun a partir del trabajo JSR 220 y que ha sido incluida en el estándar EJB3.

El objetivo que persigue el diseño de este API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos relacional siguiendo el patrón ORM (Mapeo Objeto-Relacional).

Para ello JPA realiza los siguientes aportes:

Un API en si mismo, definido en el paquete `javax.persistence` que será el interfaz que deberán implementar los distintos ORMs. Destacan las siguientes implementaciones:

- Hibernate: <http://www.hibernate.org>
- Oracle Toplink: <http://www.oracle.com/technetwork/middleware/toplink/overview/index.html>
- OpenJPA: <http://openjpa.apache.org/>
- EclipseLink: <http://www.eclipse.org/eclipselink/>
- DataNucleus (antiguo JPOX): <http://www.datanucleus.org/>

#### 3.6.6.1. Entidades

Una entidad es un objeto de la persistencia del dominio de peso ligero. Normalmente, una entidad representa una tabla en una base de datos relacional, y cada instancia de la entidad corresponde a una fila de esa tabla. El artefacto de programación principal de una entidad es la clase de entidad, si bien las entidades pueden utilizar las clases de ayuda.

El estado persistente de una entidad es representada, ya sea a través de campos persistentes o propiedades persistentes. Estos campos o propiedades se mapean mediante anotaciones, así como las relaciones entre entidades en el almacén de datos subyacente.

Requisitos para las clases de entidad:

- Debe tener la anotación de clase `javax.persistence.Entity`.
- La clase debe tener constructor sin argumentos público o protegido, aunque también puede tener otros constructores.
- La clase no debe ser declarada final. No hay métodos o variables persistentes instancia debe ser declarada `definitive`.
- Si una instancia de la entidad pueden pasar por valor, como un objeto individual, como a través de la interfaz de un *bean* de sesión de negocios a distancia, la clase debe implementar el interfaz `Serializable`.
- Las propiedades persistentes de cada entidad, deben ser declaradas privadas, protegidas o de paquete.

El estado persistente de una entidad se puede acceder ya sea a través de variables de instancia de la entidad o a través de las propiedades de JavaBeans. Los campos o propiedades pueden ser de los siguientes tipos:

- Java tipos primitivos y sus envoltorios herederos de Object correspondientes.
- `java.lang.String`
- `java.math.BigInteger`
- `java.math.BigDecimal`
- `java.util.Date`
- `java.util.Calendar`
- `java.sql.Date`
- `java.sql.Time`
- `java.sql.Timestamp`
- Tipos definidos por el usuario Serializables
- `byte []`
- `Byte []`
- `char []`
- `Character []`
- Los tipos enumerados
- Otras entidades y / o colecciones de entidades
- Clases incorporables

#### 3.6.6.2. Los campos no persistentes

Todos los campos no anotados como `@javax.persistence.Transient` y `transient` son considerados en el almacén de datos.

#### 3.6.6.3. Claves primarias

Una entidad que tenga una clave primaria en su tabla de BBDD correspondiente deberá tener una propiedad anotada como `javax.persistence.Id`

Cada entidad tiene un identificador de objeto único. Una entidad de cliente, por ejemplo, podría ser identificado por un número de cliente. El identificador único o **clave primaria**, permite a los clientes para localizar una instancia de la entidad en particular. Cada entidad debe tener una clave primaria. Una entidad puede tener una simple o una clave principal compuesta.

Simple uso de las claves principales de la `javax.persistence.Id` anotación para indicar la propiedad de la clave principal o campo.

Claves primarias compuestas debe corresponderse con una sola propiedad persistente o campo, o de un conjunto de propiedades individuales persistente o campos. Claves primarias compuestas debe ser definido en una clase de clave primaria. Claves primarias compuestas se denotan con el `javax.persistence.EmbeddedId` y `javax.persistence.IdClass` anotaciones.

La clave principal o la propiedad o campo de una clave primaria compuesta, debe ser uno de los tipos de lenguaje Java siguientes:

- Java tipos primitivos

- Java tipos de primitivas *wrapper*
- `java.lang.String`
- `java.util.Date` (el tipo debe ser temporal `DATE`)
- `java.sql.Date`

Tipos de punto flotante no se debe utilizar en las claves principales. Si utiliza una clave generada primaria, sólo los tipos integral será portátil.

#### 3.6.6.4. Multiplicidad de relaciones entre entidades

Hay cuatro tipos de multiplicidades: uno a uno, uno-a-muchos, muchos-a-uno, y muchos-a-muchos.

- **Uno a uno:** Cada instancia de la entidad está relacionada con una sola instancia de otra entidad. Por ejemplo, el modelo de un almacén físico en el que cada recipiente de almacenamiento contiene un control único, `StorageBin` y `widgets` que tienen una relación uno a uno. Para utilizar una relación Uno-a-uno hay que utilizar la anotación `javax.persistence.OneToOne` en la propiedad persistente correspondiente.
- **De uno a muchos:** Una instancia de entidad puede estar relacionada con varias instancias de otras entidades. Por ejemplo, un pedido de venta puede tener varios artículos. Las relaciones uno-a-muchos utilizan la anotación `javax.persistence.OneToMany` en la propiedad persistente correspondiente.
- **Muchos-a-uno:** Múltiples instancias de una entidad pueden estar relacionadas con una sola instancia de otra entidad. Esta multiplicidad es lo contrario de una relación uno-a-muchos. Para utilizar una relación muchos-a-uno hay que utilizar la anotación `javax.persistence.ManyToOne` en la propiedad persistente correspondiente.
- **De muchos a muchos:** las instancias de una entidad pueden estar relacionadas con varias instancias de otra. Por ejemplo, en la universidad, cada curso tiene muchos alumnos, y cada estudiante puede tomar varios cursos. Las relaciones muchos-a-muchos utilizan la anotación `javax.persistence.ManyToMany` en la propiedad persistente correspondiente.

#### 3.6.6.5. Dirección en las relaciones entre entidades

La dirección de una relación puede ser bidireccional o unidireccional. Una relación bidireccional tiene tanto un lado de la propiedad y un lado inverso. Una relación unidireccional sólo tiene una parte propietaria. La parte propietaria de una relación determina el tiempo de ejecución de persistencia hace cambios a la relación en la base de datos.

##### *Las relaciones bidireccionales*

En una relación **bidireccional** cada entidad tiene un campo de relación o la propiedad que se refiere a la otra entidad. A través del campo de relación o la propiedad, el código de una clase de entidad puede acceder a su objeto relacionado. Las relaciones bidireccionales deben seguir estas reglas:

- El lado inverso de una relación bidireccional debe poseer el elemento `mappedBy` con la anotación `@OneToOne`, `@OneToMany` o `@ManyToMany`. El elemento `mappedBy` designa la propiedad o campo en la entidad, que es el dueño de la relación.

- El lado de muchos de una relación bidireccional muchos-a-uno no debe definir el elemento `mappedBy`. El lado de muchos es siempre el lado propietario de la relación.

Para una relación bidireccional muchos-a-muchos tanto un lado como el otro de la relación puede ser la parte propietaria.

#### *Las relaciones unidireccionales*

En una relación **unidireccional**, sólo una de las entidades inmersas en la relación tiene un campo de relación o la propiedad que se refiere a la otra.

#### 3.6.6.6. Consultas y Dirección Relación

JPA-QL se usa para navegar a través de las relaciones. La dirección de una relación determina si una consulta se puede navegar de una entidad a otra

#### 3.6.6.7. Eliminaciones en cascada en entidades relacionadas

Las entidades que utilizan relaciones, a menudo, tienen dependencias en la existencia de la otra entidad en la relación.

Para eliminar en cascada las relaciones se especifican mediante la especificación del atributo `cascade=REMOVE` en la anotación `@OneToOne` y `@OneToMany`.

Por ejemplo:

```
@OneToMany (cascade = REMOVE, mappedBy = "cliente")
public Set<Order> getOrders() { return orders; }
```

#### 3.6.6.8. Herencia de entidades

Las entidades soportan herencia de clases, asociaciones polimórficas y consultas polimórficas. Las clases padres (*super classes*) pueden no ser entidades o no, siendo estas abstractas o concretas.

#### 3.6.6.9. Entidades abstractas

Una clase abstracta puede ser declarada como una entidad mediante la decoración de la clase con `@Entity`. Las entidades abstractas difieren de entidades concretas en que no pueden ser instanciadas.

Las entidades abstractas se pueden consultar igual que las consultas concretas. Si una entidad abstracta, es el objetivo de una consulta, la consulta funciona en todas las subclases concretas de la entidad abstracta.

```
@Entity
public abstract class Employee {
    @Id
    protected Integer employeeId;
    ...
}

@Entity
public class FullTimeEmployee extends Employee {
    protected Integer salary;
```

```

    ...
}

@Entity
public class PartTimeEmployee extends Employee {
    protected Float hourlyWage;
}

```

#### 3.6.6.10. Superclases mapeadas

Las entidades pueden heredar de superclases que contienen el estado persistente e información de mapas, pero no son entidades. Es decir, la superclase no está anotada como `@Entity`, y no se asigna como una entidad por el proveedor de persistencia Java. Estas superclases se utilizan con mayor frecuencia cuando se tiene el estado y la información común a varias clases de entidad.

Una superclase mapeada se especifica mediante la anotación de clase

`javax.persistence.MappedSuperclass`.

```

@MappedSuperclass
public class Employee {
    @Id
    protected Integer employeeId;
    ...
}

@Entity
public class FullTimeEmployee extends Employee {
    protected Integer salary;
    ...
}

@Entity
public class PartTimeEmployee extends Employee {
    protected Float hourlyWage;
    ...
}

```

La superclase heredada no es consultable, se deben utilizar las subclases entidad de la superclase heredada. La superclase no puede ser objeto de las relaciones entre entidades y estas puede ser abstracta o concreta.

Las superclases mapeadas no tienen tablas correspondientes en el almacén de datos subyacente. Las entidades que heredan de la superclase deben definir las asignaciones de tablas. Por ejemplo, en el ejemplo de código anterior, las tablas subyacentes serían

`FULLTIMEEMPLOYEE` y `PARTTIMEEMPLOYEE`, pero no existe la tabla `EMPLOYEE`.

#### 3.6.7. Hibernate vs. iBatis vs. JPA

Una vez se han explicado estos tres ORMs, podemos pasar a compararlos. Como ya se ha indicado JPA no es un ORM, sino la interfaz de nuestra aplicación con el ORM usado y por otro lado Hibernate e iBatis son dos ORMs muy diferentes. Si consideramos como ORM más primitivo aquel que únicamente se basa en conexiones JDBC y consultas SQL, tenemos que iBatis es más primitivo que Hibernate y por tanto, Hibernate es el ORM más potente e independiente del SGBD al que se está accediendo. Por tanto, iBatis será, a nivel de

aplicación, mucho más eficiente, pero por contraprestación, más complicado para el desarrollador. Por esas mismas circunstancias Hibernate será más rápido durante el desarrollo de aplicaciones, pero menos eficiente en ejecución que iBatis. Por dicho motivo, la elección de un ORM u otro deberá basarse en la experiencia de los desarrolladores y de sus conocimientos en SQL, ya que tener el control de las consultas que se están ejecutando en todo momento siempre será la opción más eficiente.

En esta tabla se resumen estas conclusiones:

Características	iBATIS	Hibernate	JPA
Sencillez	Mejor	Bueno	Bueno
Solución completa de ORM	Medio	Mejor	Mejor
Adaptabilidad a los cambios del modelo de datos	Bueno	Medio	Medio
Complejidad	Mejor	Medio	Medio
La dependencia de SQL	Bueno	Medio	Medio
Rendimiento	Mejor	Mejor	*
Portabilidad a través de diferentes bases de datos relacionales	Medio	Mejor	*
Portabilidad a otras plataformas distintas de Java	Mejor	Bueno	No se admite
Apoyo a la comunidad y la documentación	Medio	Bueno	Bueno

\* Las funciones admitidas por la JPA depende del proveedor de persistencia y el resultado final puede variar en consecuencia.

### 3.7. Factoría de instancias y acople de distintas capas

#### 3.7.1. Spring Framework

El **Spring Framework** (también conocido simplemente como **Spring**) es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java. La primera versión fue escrita por Rod Jonhson, quien lo lanzó primero con la publicación de su libro *Expert One-on-One Java EE Design and Development* (Wrox Press, octubre 2002). 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 al modelo de Enterprise JavaBean (EJB). Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

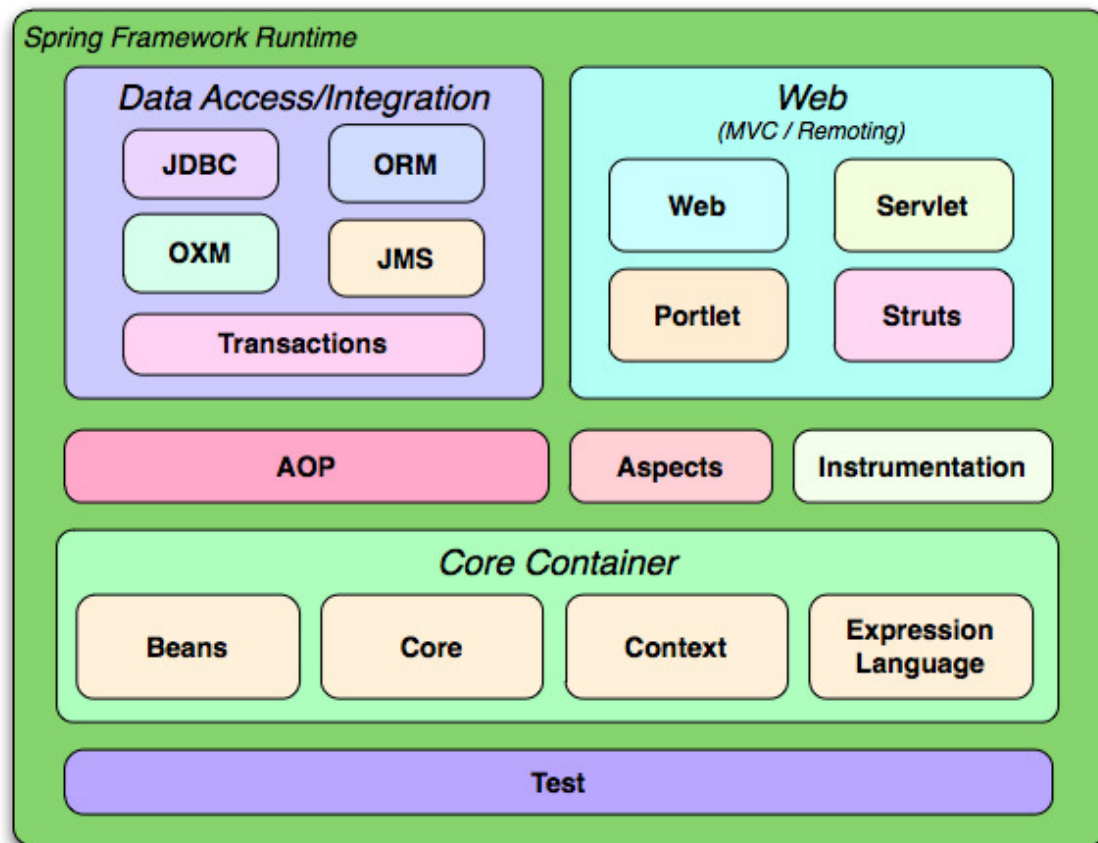
Mientras que las características fundamentales de este framework pueden emplearse en cualquier aplicación hecha en Java, existen muchas extensiones y mejoras para construir aplicaciones basadas en web por encima de la plataforma empresarial de Java (Java Enterprise Platform).

Aportes que realiza Spring frente a otros frameworks:

- **Se ocupa de aspectos importantes que otros muchos marcos populares no.** Spring se centra en ofrecer mecanismos que ayuden a gestionar los objetos de la lógica de negocio.
- **Spring es un Framework ampliable.** Spring tiene una arquitectura en capas, lo que significa que usted puede optar por utilizar casi cualquier parte de ella de forma aislada, sin embargo, su arquitectura es internamente consistente. Usted puede optar por utilizar Spring sólo para simplificar el uso de JDBC, por ejemplo, o puede optar por usar Spring para gestionar todos los objetos de negocio.
- **Spring está diseñado desde cero para ayudarle a escribir código que es fácil de probar.** Spring es un marco ideal para proyectos basado en pruebas (Test Driven Development).
- **Spring se ha convertido en un estándar como marco de integración con otros componentes.** Componentes como CAS, Acegi Security (actual Spring Security), Struts, Struts 2, JSF, etc ya contemplan su integración.

Spring está compuesto por aproximadamente 20 módulos que están agrupados en cinco contenedores básicos:

- Core container
- Spring Beans
- Spring Core
- Spring Context
- Acceso a Datos / Integración
- Spring JDBC
- Spring ORM: Integración con motores de persistencia: JPA iBatis e Hibernate
- JXM Integration
- JMS Integration
- Spring Transaction
- Web
- Spring MVC
- Spring Remoting
- Servlet Support
- Portlet Support
- Apache Struts Integration
- Java Server Faces (JSF) Integration
- Spring AOP
- Integración con AspectJ
- Spring Test



Frameworks compatibles con Spring

**CAS (Central Authentication Services):** Es un protocolo SSO (single sign-on) que se encarga de la autenticación de accesos a determinadas aplicaciones.

CAS se divide en tres componentes: Servidor de CAS, Servidor de aplicación destinataria y navegador web (cliente).

Cuenta con una serie de proveedores de autenticación ya implementados como, así como mecanismos de extensión para hacerlos a medida.

Es configurable a través de beans en el contexto de Spring.

**Acegy/Spring Security:** Proporciona servicios de seguridad para aplicaciones de software empresariales basados en J2EE y enfocado sobre proyectos construidos que usan Spring Framework:

Mecanismos de gestión de acceso seguro a determinadas URLs, métodos y clases Java, así como a instancias concretas de clases.

Permite separar la lógica de nuestras aplicaciones del control de la seguridad, utilizando filtros para las peticiones al servidor de aplicaciones o aspectos para la seguridad en clases y métodos.

La configuración de la seguridad es portable de un servidor a otro, ya que se encuentra dentro del WAR o el EAR de nuestras aplicaciones.

Soporta muchos modelos de identificación de usuarios (HTTP BASIC, HTTP Digest, basada en formulario, LDAP, OpenID, JAAS y muchos más).



**Spring-WS** (Servicios Web con Spring): Subproyecto de Spring destinado a facilitar la creación de servicios web basados en el intercambio de documentos (document driven).

Principales características:

Facilita aplicar las mejores practicas para la creación de servicios web

Facilidad para distribuir los pedidos xml a través de diferentes tipos de mapeos

Soporte para varias librerías de manejo de XML (DOM, SAX, StAX, JDOM, dom4j, XDOM)

Soporte para mapeo de xml a objetos (Castor, JiBX, JAXB, XStream)

Integración con Spring Framework

**Spring MVC:** Framework MVC de Spring. Será explicado en el siguiente epígrafe.

### 3.7.2. Spring en detalle

#### 3.7.2.1. Spring IOC (Inversión del control)

Para explicar en consiste el paradigma de la inversión del control, partamos de la ecuación siguiente:

IOC (Inversión del control) = DI (Inyección de dependencias) = Principio de Hollywood = “No nos llames, ya te llamaremos nosotros”

La inversión del control y la inyección de dependencias son dos términos que es habitual que se confundan entre sí. Esto es debido a que ambos tienen sentido de forma conjunta, que es lo que Spring nos facilita. Veamos con detalle sus definiciones:

**Inversión de control** (*Inversion of Control* en inglés, **IoC**) es un método de programación en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales, en los que la interacción se expresa de forma imperativa haciendo llamadas a procedimientos o funciones. En el modelo de programación tradicional, el programador especificaba una secuencia de decisiones y procedimientos que pueden darse durante el ciclo de vida de un programa mediante llamadas a funciones. Por el contrario, en la inversión de control se especifican respuestas deseadas a sucesos o solicitudes de datos concretas, dejando que algún tipo de entidad o arquitectura externa lleve a cabo las acciones de control que se requieran en el orden necesario y para el conjunto de sucesos que tengan que ocurrir.

En cierto modo es una implementación del Principio de Hollywood, una metodología de diseño de software, cuyo nombre proviene de las típicas respuestas que se les dan a los actores amateurs en las audiciones que tienen lugar en la meca del cine: *no nos llames; nosotros te llamaremos*.

#### 3.7.2.2. Inyección de Dependencias

En inglés *Dependency Injection*, DI. Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. El término fue acuñado por primera vez por Martin Fowler.

La forma habitual de implementar este patrón es mediante un "**Contenedor DI**" y **objetos POJO**. El contenedor inyecta a cada objeto los objetos necesarios según las relaciones plasmadas en un fichero de configuración.

Típicamente este contenedor es implementado por un framework externo a la aplicación (como Spring o uno propietario), por lo cual en la aplicación también se utilizará inversión de control al ser el contenedor (almacenado en una biblioteca) quien invoque el código de la aplicación.

En resumen, el tándem IoC-DI ofrece las siguientes mejoras:

Se consigue bajar el acoplamiento entre los distintos componentes. Cada componente es independiente de los demás, y no sabe de dónde salen los servicios que utiliza.

Se aumenta la mantenibilidad y extensibilidad de la solución. Es fácil ampliar la funcionalidad de las aplicaciones con riesgos mínimos sobre las piezas que ya están bien probadas. El impacto de modificar una pieza es mínimo en el resto de la aplicación, ya que son compartimentos bien separados.

Más fácil hacer pruebas unitarias (TDD). A cada pieza se le pueden pasar unas dependencias “dummy” (mock objects) para hacer pruebas unitarias.

Elementos básicos de Spring: Beans y contenedores:

#### Los Beans:

- Un bean es un objeto que forma parte de la aplicación y que es instanciado, configurado y gestionado por el contenedor
- No tienen dependencias con Spring
- Es un POJO que no tiene por que cumplir con la definición de JavaBean
- Los beans y sus dependencias están definidos en los metadatos del contenedor (normalmente un xml de configuración)

### 3.8. Frameworks MVC

#### 3.8.1. Spring MVC

**Spring MVC** es uno de los módulos del Framework de Spring, y como su propio nombre nos indica que implementa una arquitectura Modelo - Vista - Controlador.

Una aplicación **AppEjemplo** que usa Spring MVC debe cumplir las siguientes características:

AppEjemplo

```
src/ejemploapp/web/HelloController.java
index.jsp
WEB-INF
    classes
        ejemploapp.web.HelloController.class
    lib
        spring.jar
        spring-webmvc.jar
        commons-logging.jar
ejemploapp-servlet.xml
web.xml
```

**index.jsp:**

```

<html>
  <head><title>Example :: Spring Application</title></head>
  <body>
    <h1>Example - Spring Application</h1>
    <p>This is my test.</p>
  </body>
</html>

```

#### web.xml:

```

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

  <servlet>
    <servlet-name>ejemploapp</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>ejemploapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>
      index.jsp
    </welcome-file>
  </welcome-file-list>

</web-app>

```

#### ejemploapp-servlet.xml:

```

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd">

  <!-- the application context definition for the EjemploApp
  DispatcherServlet -->

  <bean name="/hello.htm" class="ejemploapp.web.HelloController"/>

</beans>

```

## HelloController.java

```
package ejemploapp.web;

import org.springframework.web.servlet.mvc.Controller;
import org.springframework.web.servlet.ModelAndView;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.io.IOException;

public class HelloController implements Controller {

    protected final Log logger = LogFactory.getLog(getClass());

    public ModelAndView handleRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {

        logger.info("Returning hello view");
        return new ModelAndView("hello.jsp");
    }
}
```

## HelloControllerTest.java

```
package ejemploapp.web;

import org.springframework.web.servlet.ModelAndView;
import ejemploapp.web.HelloController;
import junit.framework.TestCase;

public class HelloControllerTest extends TestCase {

    public void testHandleRequestView() throws Exception{
        HelloController controller = new HelloController();
        ModelAndView modelAndView = controller.handleRequest(null,
null);
        assertEquals("hello.jsp", modelAndView.getViewName());
    }
}
```

### 3.8.2. Apache Struts 2

#### Introducción

El desarrollo de aplicaciones J2EE se viene realizando desde 1997, cuando Java publicó su especificación de Servlets. En el año 2000 surgió Apache Struts con el objetivo de facilitar el desarrollo de aplicaciones web y se convirtió en un estándar de facto en el desarrollo de aplicaciones web durante varios años y coincidiendo con el *boom* de las empresas *punto com*.



Apache Struts 2 es una evolución de Apache Struts que adopta, a su vez, funcionalidades de otro framework MVC, Opensymphony XWork. Pretende reducir el nivel de acoplamiento de su predecesor y simplificar aún más el desarrollo de aplicaciones web proporcionando mecanismos de reutilización de código en las vistas como ya comentaremos más adelante.

La utilización de XWork no es meramente conceptual, se puede comprobar fácilmente revisando las librerías de una aplicación Struts 2:

```
struts2-core.jar  
xwork-core.jar
```

#### Apache Struts 2: Características

Hay muchos frameworks MVC, pero pocos tenían las siguientes características en su momento de creación:

- Framework basado en acciones (actions)
- Configuración mediante anotaciones o un fichero XML
- Acciones basadas en POJO más simples de probar
- Integración con Spring, SiteMesh y Apache Tiles.
- Integración con OGNL
- Temas de presentación basados en librerías de etiquetas (tags)
- Etiquetas para AJAX
- Múltiples opciones para la capa de vista (JSP, Freemarker, Velocity y XSLT)
- Framework extensible y modificable a través de plugins.

Veamos a grandes rasgos el funcionamiento de Struts2 como MVC:

**M (modelo):** Son las acciones, el bean con métodos `getXx()` y `setXx()` que contiene el estado de la acción y la ejecución

**V (vista):** Serán las JSPs que se renderizaran al presentar la información resultante al navegador invocante.

**C (controlador):** En el caso de este framework es muy simple. Se corresponde con un filtro despachador implementado por Struts y una pila de interceptores que determinarán el ciclo de vida de cada acción.

### 3.8.2.1. Arquitectura de Struts 2

Struts 2 Framework ofrece una interfaz que permite extender su comportamiento estandar de una forma elegante para el desarrollo de aplicaciones Web empresariales de cualquier tamaño.

#### Ciclo de vida de una petición en una aplicación Struts 2

**Usuario envía la solicitud:** El usuario envía una solicitud al servidor de algún recurso.

**El filtro FilterDispatcher determina la acción apropiada.**

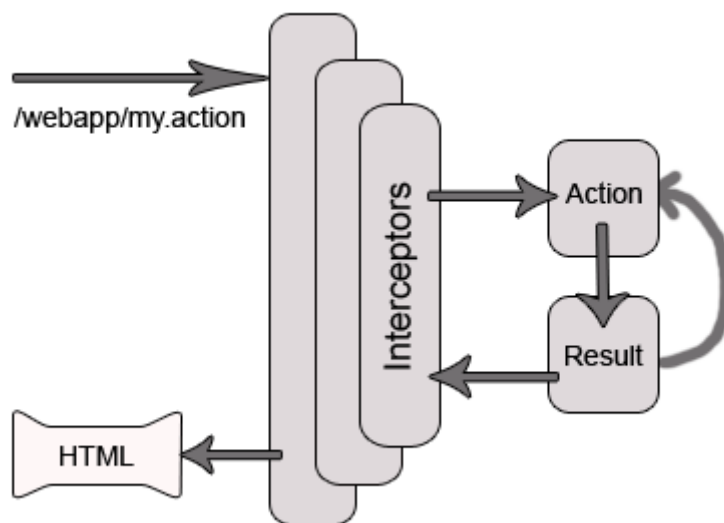
**Se ejecutan los interceptores:** Se ejecutan los interceptores configurados en la aplicación que se encargan de ofrecer las funcionalidades comunes, como el flujo de trabajo, validación, carga de archivos, etc.

**Ejecución de la acción:** A continuación, se ejecuta el método de la acción para realizar las operaciones propias de la acción. Como pueden ser operaciones de bases de datos de almacenamiento o recuperación de datos, etc.

**Presentación de salid:** Se renderiza la salida.

**Devolución de la solicitud:** A continuación, la solicitud vuelve a través de los interceptores en el orden inverso.

**Mostrar el resultado para el usuario:** Por último, el control se devuelve al contenedor de servlets, que envía la salida al navegador del usuario.



### Struts 2 descripción de alto nivel de procesamiento de la solicitud

#### 3.8.2.2. Struts 2 Arquitectura

Struts 2 es un marco frontal del controlador muy elegante y flexible sobre la base de muchas tecnologías estándar como filtros de Java, Java Beans, ResourceBundle, XML, etc

Para el **modelo**, se puede utilizar cualquier tecnología de acceso a datos, como JDBC, EJB, Hibernate, etc y para la **vista**, el marco puede ser integrado con JSP, Apache Tiles, plantillas Velocity, Sitemesh, PDF, XSLT, etc.

#### 3.8.2.3. Manejo de excepciones:

El marco de trabajo Struts 2 nos permite definir los controladores de excepciones y interceptores.

**Manejadores de excepciones:** El manejo de excepciones, nos permite definir el procedimiento de manejo de excepciones en el nivel global y local. Marco detecta la

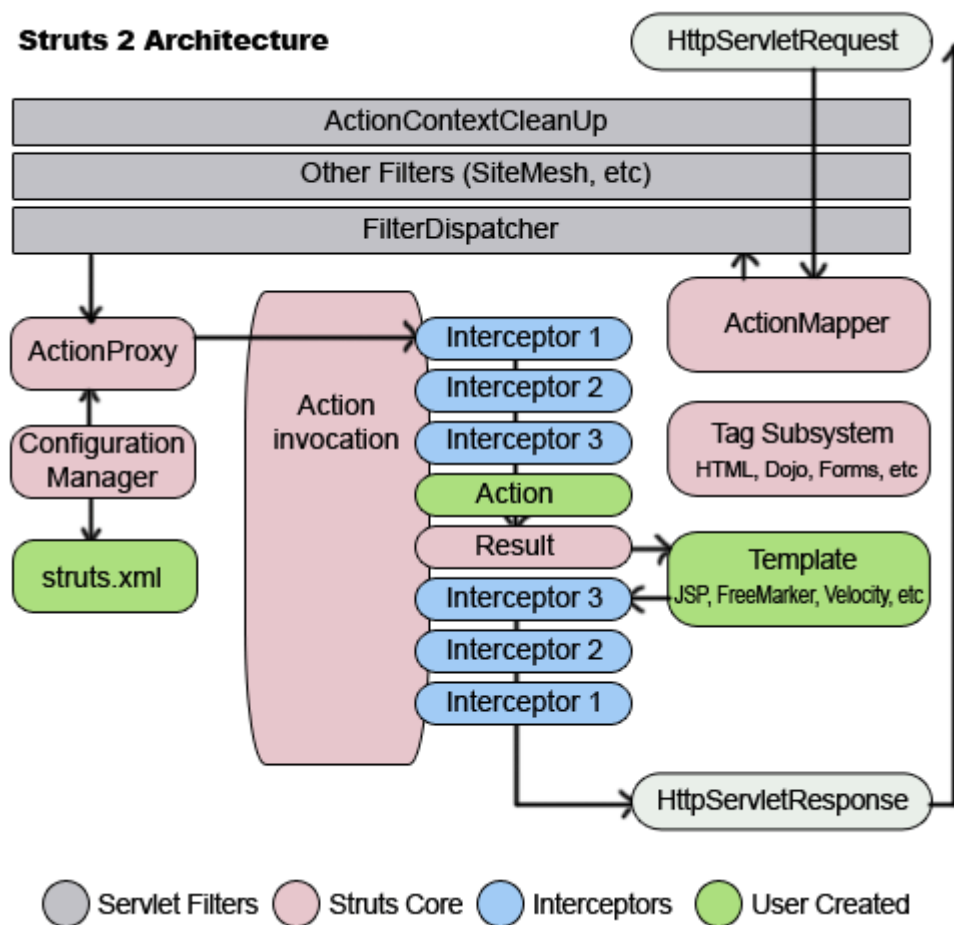
excepción y, a continuación se muestra la página de nuestra elección con el mensaje apropiado y detalles de la excepción.

### Los interceptores:

Los interceptores se utilizan para especificar la "solicitud de procesamiento del ciclo de vida" para una acción. Los interceptores están configurados para aplicar las funcionalidades comunes como el flujo de trabajo, validación de la solicitud, etc.

El siguiente diagrama muestra la arquitectura de Struts 2 Marco y también muestra la solicitud inicial va al contenedor de servlets, como Tomcat, que luego se pasa a través de la cadena de archivador estándar.

### Arquitectura:



La cadena de filtros incluye:

- **FilterDispatcher:** El filtro **FilterDispatcher** utiliza el **ActionMapper** para determinar si se debe invocar una acción o no. Si la acción se requiere que se invoque, el **FilterDispatcher** cede el control al **ActionProxy**.
- **ActionProxy:** El **ActionProxy** revisa la configuración de la aplicación a través del archivo **struts.xml**. A continuación, el **ActionProxy** crea un **ActionInvocation**, que implementa el patrón comando lanzando un proceso en el que invoca a los interceptores y luego llama a la acción. Una vez se obtiene el resultado, se pasa a la parte de presentación de la capa de vista (JSP, plantillas, etc) volviéndose a ejecutar los interceptores en orden inverso.

### 3.8.2.4. El fichero struts.xml

El marco de trabajo Struts 2 utiliza un archivo de configuración (`struts.xml`) para iniciar sus propios recursos. Estos recursos incluyen:

- Los interceptores que pueden preprocesar y postprocesar una solicitud
- Clases de acción que pueden llamar a la lógica de negocio y el código de acceso a datos
- Los resultados que se pueden preparar en la capa de vista usando JSP, plantillas Velocity y FreeMarker.
- Durante la ejecución, hay una sola configuración de una aplicación. Hay varios elementos que pueden ser configurados, incluyendo los paquetes, espacios de nombres que incluyen acciones, resultados, interceptores y manejadores de excepciones.

El archivo **struts.xml** es el archivo de configuración básica para el marco y que deben estar presente en el *classpath* de la aplicación web:

Veamos su estructura con una aplicación de ejemplo HelloWorld:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<constant name="struts.enable.DynamicMethodInvocation" value="false"
/>
<constant name="struts.devMode" value="true" />

<package name="crudfw" namespace="/crudfw" extends="struts-default">

  <action name="HelloWorld"
    class="org.crudframework.Struts2HelloWorld">
    <result>/pages/HelloWorld.jsp</result>
  </action>

  <!-- Add actions here -->
</package>

<!-- Add packages here -->
</struts>
```

**Para mostrar toda la capacidad de expresión de este fichero, basta con mostrar el DTD (Document Type Definition) <http://struts.apache.org/dtds/struts-2.0.dtd>**

:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT struts (package|include|bean|constant)*>

<!ELEMENT package (result-types?, interceptors?, default-
interceptor-ref?, default-action-ref?, default-class-ref?,
global-results?, global-exception-mappings?, action*)>
```



```
<!--ATTLIST package
    name CDATA #REQUIRED
    extends CDATA #IMPLIED
    namespace CDATA #IMPLIED
    abstract CDATA #IMPLIED
    externalReferenceResolver NMTOKEN #IMPLIED>

<!--ELEMENT result-types (result-type+)>

<!--ELEMENT result-type (param*)>
<!--ATTLIST result-type
    name CDATA #REQUIRED
    class CDATA #REQUIRED
    default (true|false) "false">

<!--ELEMENT interceptors (interceptor|interceptor-stack)+>

<!--ELEMENT interceptor (param*)>
<!--ATTLIST interceptor
    name CDATA #REQUIRED
    class CDATA #REQUIRED>

<!--ELEMENT interceptor-stack (interceptor-ref*)>
<!--ATTLIST interceptor-stack
    name CDATA #REQUIRED>

<!--ELEMENT interceptor-ref (param*)>
<!--ATTLIST interceptor-ref
    name CDATA #REQUIRED>

<!--ELEMENT default-interceptor-ref (#PCDATA)>
<!--ATTLIST default-interceptor-ref
    name CDATA #REQUIRED>

<!--ELEMENT default-action-ref (#PCDATA)>
<!--ATTLIST default-action-ref
    name CDATA #REQUIRED>
```

```
<!ELEMENT default-class-ref (#PCDATA)>
<!ATTLIST default-class-ref
    class CDATA #REQUIRED>

<!ELEMENT global-results (result+)>

<!ELEMENT global-exception-mappings (exception-mapping+)>

<!ELEMENT action (param|result|interceptor-ref|exception-
mapping)*>
<!ATTLIST action
    name CDATA #REQUIRED
    class CDATA #IMPLIED
    method CDATA #IMPLIED
    converter CDATA #IMPLIED>

<!ELEMENT param (#PCDATA)>
<!ATTLIST param
    name CDATA #REQUIRED>

<!ELEMENT result (#PCDATA|param)*>
<!ATTLIST result
    name CDATA #IMPLIED
    type CDATA #IMPLIED>

<!ELEMENT exception-mapping (#PCDATA|param)*>
<!ATTLIST exception-mapping
    name CDATA #IMPLIED
    exception CDATA #REQUIRED
    result CDATA #REQUIRED>

<!ELEMENT include (#PCDATA)>
<!ATTLIST include
    file CDATA #REQUIRED>

<!ELEMENT bean (#PCDATA)>
<!ATTLIST bean
    type CDATA #IMPLIED
```

```

    name CDATA #IMPLIED
    class CDATA #REQUIRED
    scope CDATA #IMPLIED
    static CDATA #IMPLIED
    optional CDATA #IMPLIED>

<!ELEMENT constant (#PCDATA)>
<!ATTLIST constant
    name CDATA #REQUIRED
    value CDATA #REQUIRED>

```

## Explorando struts.xml

La **etiqueta de <struts>** es la etiqueta raíz de la **struts.xml**. Puede contener los siguientes tags: package, incluye, frijol y constante.

### 3.8.2.4.1. La etiqueta del paquete:

Los paquetes son una forma de agrupar acciones, resultados, tipos de resultados e interceptores. Conceptualmente, los paquetes son similares a los objetos que pueden ser extendidos y tienen las partes individuales que se pueden reemplazar por subpaquetes.

Atributo Obligatorio		Descripción
name	sí	clave para otros paquetes para hacer referencia a
extends	no	Hereda el comportamiento del paquete del que extiende
namespace	no	Proporciona una asignación de la dirección URL para el paquete.
abstract	no	Declara un paquete como abstracto (no requiere configuraciones de acción en el paquete)

### 3.8.2.4.2. La etiqueta de inclusión:

El <include /> etiqueta se utiliza para modularizar una aplicación Struts2 que necesita para incluir otros archivos de configuración:

```

<struts>
<include file="invoices-config.xml" />
<include file="admin-config.xml" />
<include file="reports-config.xml" />

```

```
</struts>
```

Hay algunos archivos que se incluyen de manera implícita. Estos son los archivos **struts-default.xml** y el **struts-plugin.xml**. Contiene tanto configuraciones predeterminadas para los tipos de resultados, interceptores, paquetes de pilas de interceptores, así como la información de configuración para el entorno de ejecución de la aplicación web (que también se puede configurar en el archivo **struts.properties**). El fichero **struts-default.xml** proporciona la configuración básica de Struts2 y el fichero **struts-plugin.xml** proporciona configuraciones para plugins. Cada plugin (empaquetado JAR) debe contener dicho archivo.

#### 3.8.2.4.3. La etiqueta de bean

La mayoría de las aplicaciones no se necesita extender la configuración de Bean. La clase Java del bean se especifica a través del atributo class.

Atributo Necesario		Descripción
class	sí	el nombre de la clase bean
type	no	la interfaz principal de esta clase de Java implementa
name	no	el nombre único de este grano, debe ser único entre los beans de otros que especifican el mismo type
scope	no	el ámbito del bean, puede ser default, singleton, request, session, thread

#### Ejemplo:

```
<struts>
  <bean type="org.crudfw.ObjectFactory" name="factory"
    class="org.crudfw.MyObjectFactory" />
  ...
</ struts>
```

### 3.8.3. JSF (JavaServer Faces)

La tecnología JSF es un marco de trabajo de interfaces de usuario del lado de servidor para aplicaciones Web basadas en tecnología Java.

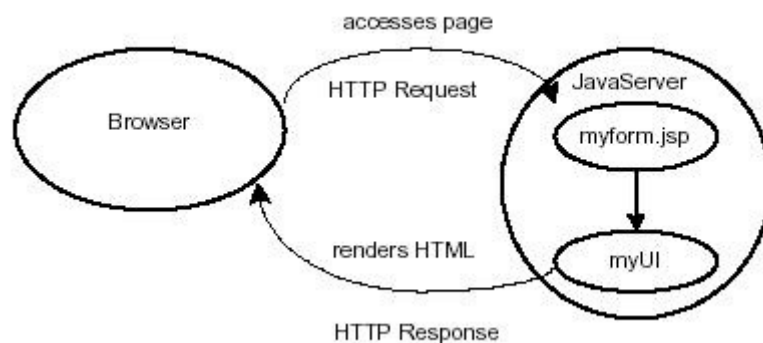
Los principales componentes de la tecnología JSF son:

- Un API y una implementación de referencia para: representar componentes UI y manejar su estado; manejo de eventos, validación del lado del servidor y conversión de datos; definir la navegación entre páginas; soportar internacionalización y accesibilidad; y proporcionar extensibilidad para todas estas características.
- Una librería de etiquetas JavaServer Pages (JSP) personalizadas para dibujar componentes UI dentro de una página JSP.

Este modelo de programación bien definido y la librería de etiquetas para componentes UI facilitan de forma significativa la tarea de la construcción y mantenimiento de aplicaciones Web con UIs del lado del servidor. Con un mínimo esfuerzo, podemos:

- Conectar eventos generados en el cliente a código de la aplicación en el lado del servidor.
- Mapear componentes UI a una página de datos del lado del servidor.
- Construir un UI con componentes reutilizables y extensibles.
- Grabar y restaurar el estado del UI más allá de la vida de las peticiones de servidor.

Como se puede apreciar en la siguiente figura, el interface de usuario que creamos con la tecnología JSF (representado por **myUI** en el gráfico) se ejecuta en el servidor y se renderiza en el cliente.



La página JSP, **myform.jsp**, dibuja los componentes del interface de usuario con etiquetas personalizadas definidas por la tecnología JavaServer Faces. El UI de la aplicación Web (representado por **myUI** en la imagen) maneja los objetos referenciados por la página JSP:

- Los objetos componentes que mapean las etiquetas sobre la página JSP.
- Los oyentes de eventos, validadores, y los conversores que está registrados en los componentes.
- Los objetos del modelo que encapsulan los datos y las funcionalidades de los componentes específicos de la aplicación.

#### 3.8.3.1 Beneficios de la Tecnología JavaServer Faces

Una de las grandes ventajas de la tecnología JavaServer Faces es que ofrece una clara separación entre el comportamiento y la presentación. Las aplicaciones Web construidas con

tecnología JSP conseguían parcialmente esta separación. Sin embargo, una aplicación JSP no puede mapear peticiones HTTP al manejo de eventos específicos de los componentes o manejar elementos UI como objetos con estado en el servidor. La arquitectura de JSF nos permite construir aplicaciones Web que implementan una separación entre el comportamiento y la presentación tradicionalmente ofrecidas por la arquitectura UI del lado del cliente.

### 3.8.3.2 Ejemplo

Aquí tenemos la clase `UserNumberBean.java` que contiene los datos introducidos en el campo de texto de la página `greeting.jsp`:

```
package guessNumber;
import java.util.Random;

public class UserNumberBean {
    Integer userNumber = null;
    Integer randomInt = null;
    String response = null;

    public UserNumberBean () {
        Random randomGR = new Random();
        randomInt = new Integer(randomGR.nextInt(10));
        System.out.println("Duke's Number: "+randomInt);
    }

    public void setUserNumber(Integer user_number) {
        userNumber = user_number;
        System.out.println("Set userNumber " + userNumber);
    }

    public Integer getUserNumber() {
        System.out.println("get userNumber " + userNumber);
        return userNumber;
    }

    public String getResponse() {
        if(userNumber.compareTo(randomInt) == 0)
            return "Yay! You got it!";
        else
            return "Sorry, "+userNumber+" is incorrect.";
    }
}
```

Como podemos ver, este *bean* es como cualquier otro componente JavaBeans. Tiene un método `set()` o accesor y un campo privado o propiedad. Esto significa que podemos concebir referenciar beans que ya hayamos escrito desde nuestras páginas JSF.

Dependiendo del tipo de componente que referencia una propiedad del objeto del modelo, esta propiedad puede ser de cualquiera de los tipos básicos primitivos y los tipos referencia. Esto incluye cualquiera de los tipos numéricos, `String`, `int`, `double`, y `float`. JSF se encargará de convertir el dato al tipo especificado por la propiedad del objeto del modelo.

También podemos aplicar una conversión a un componente para convertir los valores de los componentes a un tipo que no esté soportado por el componente.

### 3.8.3.3 Modelo (Bean controlado)

Después de desarrollar los *beans* utilizados en la aplicación, necesitamos añadir declaraciones para ellos en el fichero de configuración de la aplicación:

```
<managed-bean>
  <managed-bean-name>UserNumberBean</managed-bean-name>
  <managed-bean-class>
    guessNumber.UserNumberBean
  </managed-bean-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

La implementación de JavaServer Faces procesa este fichero en el momento de arrancada de la aplicación e inicializa el `UserNumberBean` y lo almacena en el ámbito de sesión. Entonces el bean estará disponible para todas las páginas de la aplicación.

### 3.8.3.4 Vista (Página JSF)

Las páginas contendrán los componentes UI, los mapeos entre los componentes y los datos de los objetos del modelo, y otras etiquetas importantes (como etiquetas del validador) a las etiquetas de los componentes. Aquí tenemos la página `greeting.jsp` con las etiquetas de validador (menos los HTML que lo rodea):

```
<HTML>
<HEAD> <title>Hello</title> </HEAD>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<body bgcolor="white">
<h:graphic_image id="wave_img" url="/wave.med.gif" />
<h2>Hi. My name is Duke.
I'm thinking of a number from 0 to 10.
Can you guess it?</h2>
<f:useFaces>
  <h:form id="helloForm" formName="helloForm" >
    <h:graphic_image id="wave_img" url="/wave.med.gif" />
    <h:input_number id="userNo" numberStyle="NUMBER"
      valueRef="UserNumberBean.userNumber">
      <f:validate_longrange minimum="0" maximum="10" />
    </h:input_number>
    <h:command_button id="submit" action="success"
      label="Submit" commandName="submit" /><p>
    <h:output_errors id="errors1" for="userNo"/>
  </h:form>
</f:useFaces>
```

Esta página demuestra unas cuantas características importantes que utilizaremos en la mayoría de nuestras aplicaciones JavaServer Faces:

### 3.8.3.5 Controlador (Navegación)

Otra posibilidad que tiene el desarrollador de la aplicación es definir la navegación de páginas por la aplicación, como qué página va después de que el usuario pulse un botón para enviar un formulario.

El desarrollador de la aplicación define la navegación por la aplicación mediante el fichero de configuración, el mismo fichero en el que se declararon los beans manejados. Aquí están las reglas de navegación definidas para el ejemplo `guessNumber`:

```

<navigation-rule>
  <from-tree-id>/greeting.jsp</from-tree-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-tree-id>/response.jsp</to-tree-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-tree-id>/response.jsp</from-tree-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-tree-id>/greeting.jsp</to-tree-id>
  </navigation-case>
</navigation-rule>

```

Cada regla de navegación define cómo ir de una página (especificada en el elemento `from-tree-id`) a otras páginas de la aplicación. El elemento `navigation-rule` puede contener cualquier número de elementos `navigation-case`, cada uno de los cuales define la página que se abrirá luego (definida por `to-tree-id`) basándose en una salida lógica (definida mediante `from-outcome`).

La salida se puede definir mediante el atributo `action` del componente `UICommand` que envía el formulario, como en el ejemplo `guessNumber`:

```

<h:command_button id="submit" action="success" label="Submit" />

```

### 3.8.3.6 El Ciclo de Vida de una Página JavaServer Faces

El ciclo de vida de una página JSF es similar al de una página JSP: El cliente hace una petición HTTP de la página y el servidor responde con la página traducida a HTML. Sin embargo, debido a las características extras que ofrece la tecnología JSF, el ciclo de vida proporciona algunos servicios adicionales mediante la ejecución de algunos pasos extras.

### 3.8.3.7 Ciclo de Vida Estándar de Procesamiento de Peticiones

La mayoría de los usuarios de la tecnología JSF no necesitarán conocer a fondo el ciclo de vida de procesamiento de una petición. Sin embargo, conociendo lo que la tecnología JSF realiza para procesar una página, un desarrollador de aplicaciones JSF no necesitará preocuparse de los problemas de renderizado asociados con otras tecnologías UI. Un ejemplo sería el cambio de estado de los componentes individuales. Si la selección de un componente como un `checkbox` afecta a la apariencia de otro componente de la página, la tecnología JSF manejará este evento de la forma apropiada y no permitirá que se dibuje la página sin reflejar este cambio.

La siguiente figura ilustra los pasos del ciclo de vida petición-respuesta JSF





UIComponent. El conjunto de clases de componentes UI incluido en la última versión de JavaServer Faces es:

- `UICommand`: Representa un control que dispara actions cuando se activa.
- `UIForm`: Encapsula un grupo de controles que envían datos de la aplicación. Este componente es análogo a la etiqueta **form** de HTML.
- `UIGraphic`: Muestra una imagen.
- `UIInput`: Toma datos de entrada del usuario. Esta clase es una subclase de `UIOutput`.
- `UIOutput`: Muestra la salida de datos en un página.
- `UIPanel`: Muestra una tabla.
- `UIParameter`: Representa la sustitución de parámetros.
- `UISelectItem`: Representa un sólo ítem de un conjunto de ítems.
- `UISelectItems`: Representa un conjunto completo de ítems.
- `UISelectBoolean`: Permite a un usuario seleccionar un valor booleano en un control, seleccionándolo o deseleccionándolo. Esta clase es una subclase de `UIInput`.
- `UISelectMany`: Permite al usuario seleccionar varios ítems de un grupo de ítems. Esta clase es una subclase de `UIInput`.
- `UISelectOne`: Permite al usuario seleccionar un ítem de un grupo de ítems. Esta clase es una subclase de `UIInput`.

#### 3.8.3.10 El Modelo de Renderizado de Componentes

La arquitectura de componentes JavaServer Faces está diseñada para que la funcionalidad de los componentes se defina mediante las clases de componentes, mientras que el renderizado de los componentes se puede definir mediante un renderizador separado. Este diseño tiene varios beneficios:

- Los escritores de componentes pueden definir sólo una vez el comportamiento de un componente, pero pueden crear varios renderizadores, cada uno de los cuales define una forma diferente de dibujar el componente para el mismo cliente o para diferentes clientes.
- Los autores de páginas y los desarrolladores de aplicaciones pueden modificar la apariencia de un componente de la página seleccionando la etiqueta que representa la combinación componente/renderizador apropiada.

Un kit renderizador define como se mapean las clases de los componentes a las etiquetas de componentes apropiadas para un cliente particular. La implementación JavaServer Faces incluye un `RenderKit` estándar para renderizar a un cliente HTML.

Por cada componente UI que soporte un `RenderKit`, éste define un conjunto de objetos `Renderer`. Cada objeto `Renderer` define una forma diferente de dibujar el componente particular en la salida definida por el `RenderKit`.

La parte `command` de las etiquetas corresponde con la clase `UICommand`, y especifica la funcionalidad, que es disparar un action. Las partes del botón y el hipervínculo de las etiquetas corresponden a un renderizador independiente, que define cómo dibujar el componente.

#### 3.8.3.11 Modelo de Conversión

Una aplicación JavaServer Faces opcionalmente puede asociar un componente con datos del objeto del modelo del lado del servidor. Este objeto del modelo es un componente `JavaBeans` que encapsula los datos de un conjunto de componentes. Una aplicación obtiene y configura

los datos del objeto modelo para un componente llamando a las propiedades apropiadas del objeto modelo para ese componente.

Cuando un componente se une a un objeto modelo, la aplicación tiene dos vistas de los datos del componente: la vista **modelo** y la vista **presentación**, que representa los datos de un forma que el usuario pueda verlos y modificarlos.

Una aplicación JavaServer Faces debe asegurarse que los datos del componente puedan ser convertidos entre la vista del modelo y la vista de presentación. Esta conversión normalmente la realiza automáticamente el renderizador del componente.

En algunas situaciones, podríamos querer convertir un dato de un componente a un tipo no soportado por el renderizador del componente. Para facilitar esto, la tecnología JavaServer Faces incluye un conjunto de implementaciones estándar de `Converter` que nos permite crear nuestros conversores personalizados. La implementación de `Converter` convierte los datos del componente entre las dos vistas.

#### 3.8.3.12 Modelo de Eventos y Oyentes

Un objetivo de la especificación JavaServer Faces es mejorar los modelos y paradigmas existentes para que los desarrolladores se puedan familiarizar rápidamente con el uso de JavaServer Faces en sus aplicaciones. En este espíritu, el modelo de eventos y oyentes de JavaServer Faces mejora el diseño del modelo de eventos de JavaBeans, que es familiar para los desarrolladores de GUI y de aplicaciones Web.

Al igual que la arquitectura de componentes JavaBeans, la tecnología JavaServer Faces define las clases `Listener` y `Event` que una aplicación puede utilizar para manejar eventos generados por componentes UI. Un objeto `Event` identifica al componente que lo generó y almacena información sobre el propio evento. Para ser notificado de un evento, una aplicación debe proporcionar una implementación de la clase `Listener` y registrarla con el componente que genera el evento.

Un ejemplo es seleccionar un checkbox, que resulta en que el valor del componente ha cambiado a `true`. Los tipos de componentes que generan estos eventos son los componentes `UIInput`, `UISelectOne`, `UISelectMany`, y `UISelectBoolean`.

Este tipo de eventos sólo se dispara si no se detecta un error de validación.

Un evento *action* ocurre cuando el usuario pulsa un botón o un hipervínculo. El componente `UICommand` genera este evento.

#### 3.8.3.13 Modelo de Validación

La tecnología JavaServer Faces soporta un mecanismo para validar el dato local de un componente durante la fase del **Proceso de Validación**, antes de actualizar los datos del objeto modelo.

Al igual que el modelo de conversión, el modelo de validación define un conjunto de clases estándar para realizar chequeos de validación comunes. La librería de etiquetas `jsf-core` también define un conjunto de etiquetas que corresponden con las implementaciones estándar de `Validator`.

La mayoría de las etiquetas tienen un conjunto de atributos para configurar las propiedades del validador, como los valores máximo y mínimo permitidos para el dato del componente. El autor de la página registra el validador con un componente anidando la etiqueta del validador dentro de la etiqueta del componente.

Al igual que el modelo de conversión, el modelo de validación nos permite crear nuestras propias implementaciones de `Validator` y la etiqueta correspondiente para realizar validaciones personalizadas.

#### 3.8.3.14 Creación del Bean Controlado

Otra función crítica de las aplicaciones Web es el manejo apropiado de los recursos. Esto incluye la separación de la definición de objetos componentes UI de los objetos de datos y almacenar y manejar estos ejemplares de objetos en el ámbito apropiado. Las versiones anteriores de la tecnología JavaServer Faces nos permitían crear objetos del modelo que encapsulaban los datos y la lógica del negocio separadamente de los objetos de componentes UI y almacenarlos en un ámbito particular. La nueva versión especifica completamente cómo se crean y se manejan estos objetos.

Esta versión presenta APIs para:

- Evaluar una expresión que se refiere a un objeto del modelo, una propiedad de un objeto del modelo, u otro tipo de datos primitivo o estructura de datos. Esto se hace con el API `Value-Binding`.
- Recuperar un objeto desde el ámbito. Esto se hace con el API `VariableResolver`.
- Crear un objeto y almacenarlo en un ámbito si no está ya allí. Esto se hace con el `VariableResolver` por defecto, llamada la **Facilidad Bean Controlado**, que se configura con el fichero del configuración de la aplicación descrito en la siguiente página.

#### 3.8.3.15 Configuración de la Aplicación

En las páginas anteriores hemos visto los distintos recursos disponibles para una aplicación JSF. Esto incluye, conversores, validadores, componentes, objetos del modelo, acciones, manejadores de navegación y otros. Los recursos de nuestra aplicación se configuran a través del fichero `faces-context.xml`

Para conocer en detalle la estructura de este fichero, veamos su DTD: [http://java.sun.com/dtd/web-facesconfig\\_1\\_1.dtd](http://java.sun.com/dtd/web-facesconfig_1_1.dtd)

### 3.8.4 GWT (Google Web Toolkit)

Google Web Toolkit (GWT) es un framework de desarrollo en Java de código abierto desarrollado por Google que surge como revulsivo para facilitar el desarrollo de aplicaciones Ajax.

Antes de nada, debemos tener claro lo que es Ajax:

**Ajax** (*Asynchronous JavaScript And XML*) (JavaScript asíncrono y XML), es una técnica de desarrollo web para crear aplicaciones interactivas. Estas aplicaciones se ejecutan en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre las páginas sin necesidad de recargarlas, lo que significa aumentar la interactividad, velocidad y usabilidad en las aplicaciones.

Los frameworks anteriores a GWT (JSF, Struts 2, Spring MVC, etc) no cuentan con mecanismos Ajax por defecto, necesitan de librerías externas como DWR, Ajax4Jsf, etc lo que hace de esta funcional un tanto tediosa y complicada.

La arquitectura de GWT se pensó para evitar este problema. Todas y cada una de las operaciones que se invocan en la capa de presentación son llamadas Ajax. Esto se consigue mediante la utilización de llamadas RPC, desde el navegador en el que se ejecuta la aplicación GWT y el servidor y utilización de eventos para capturar las respuestas de los servicios asíncronos.

GWT adopta una de las ideas que introdujo JSF en sus postulados, la de separar claramente presentación y lógica de negocio. Con la diferencia de que GWT lo hace de forma radical, convirtiendo en Javascript la parte de código relativa a presentación y comunicando ambas capas mediante llamadas RPC.

Otra característica en la que destaca GWT es que todo el código es Java. El desarrollador programa tanto la capa de presentación como la parte de lógica en código Java siguiendo un enfoque muy similar al de el desarrollo de aplicaciones Swing o AWT. Por tanto, al compilar la aplicación, la capa de presentación especificada en código Java es traducida a HTML con Javascript embebido.

En el caso de que el código Javascript de la capa de presentación no cumpla con alguna funcionalidad que tradicionalmente implementaríamos con código Javascript en nuestras JSPs o XHTMLs, GWT proporciona la posibilidad de utilizar JSNI (JavaScript Native Interface).

#### 3.8.4.1 La arquitectura de Google Web Toolkit

GWT tiene cuatro componentes principales: un compilador Java-a-JavaScript, un navegador web "hosted", y dos librerías de clases:

Los componentes son:

- **Compilador GWT Java-to-JavaScript**

El Compilador GWT Java-to-JavaScript traduce del lenguaje de programación Java a JavaScript. El compilador se utiliza cuando necesites correr tu aplicación en [modo web](#).

- **Navegador web “Hosted” de GWT**

El Navegador web “Hosted” de GWT te permite ejecutar aplicaciones GWT en modo hosted lo que estás ejecutando son bytecodes de Java sobre una máquina virtual sin compilarlos a JavaScript. Para lograr esto, el navegador GWT incrusta un controlador de browser especial (un control del Internet Explorer sobre Windows o un control de Gecko/Mozilla sobre Linux) con hooks dentro de la máquina virtual de Java.

- **Emulación de librerías JRE**

GWT contiene implementaciones en JavaScript de las librerías de clases más usadas en Java, incluyendo la mayoría de las clases del paquete **java.lang** y un subconjunto de clases del paquete **java.util**. El resto del estándar de librerías de Java no es soportado nativamente con GWT. Por ejemplo, las clases de los paquetes como **java.io** no se utilizan en aplicaciones web ya que estas acceden a recursos en la red y al sistema de archivos local.

- **Librería de clases de interfaz de usuario de GWT**

Las librerías de clases de interfaz de usuario de GWT son un conjunto de interfaces y clases personalizadas que te permiten crear **widgets** para el navegador, como botones, cajas de texto, imágenes, y texto. Éste es el núcleo de las librerías de interfaz de usuario para crear aplicaciones GWT.

Al ser código libre, podemos ver cómo está hecho GWT aquí:

<http://code.google.com/p/google-web-toolkit/>

### 3.8.5 Comparativa principales Frameworks MVC: Struts 2, JSF, y GWT

Comparar Struts 2, JSF y GWT equivale a mostrar la evolución de los frameworks J2EE de los últimos años. Para poder analizar cada framework, veamos sus Pros y sus Contras. En mi opinión, el más completo y robusto de todos es GWT.

#### 3.8.5.1 Pros y contras de Struts 2

##### Pros

- Arquitectura sencilla y fácil de extender
- Las librerías de TAGs son fáciles de adaptar desde las plantillas FreeMarker o Velocity.

##### Contras

- Documentación deficiente y mal organizada
- Demasiadas novedades frente a la versión anterior

#### 3.8.5.2 Pros y contras de JSF

##### Pros

- Permite separar claramente el contenido de la presentación y de la lógica
- Es una especificación, lo que permite tener varias implementaciones
- Gran cantidad de componentes y librerías
- Permite modificar el código de la aplicación sin conocer el lenguaje en el que está implementado
- Gran velocidad y facilidad de desarrollo

##### Contras

- La creación de componentes propios es compleja
- Requiere Javascript
- No existe una única fuente para la implementación
- No funciona bien con REST o con la gestión de la seguridad

#### 3.8.5.3 Pros y contras de GWT

##### Pros

- Gran similitud con Swing y AWT. Los desarrolladores con experiencia en estas tecnologías verán reducida notablemente la curva de aprendizaje
- Aumenta el procesamiento en el cliente frente a otros frameworks, reduciéndose las comunicaciones con el servidor
- Framework en continua evolución. Google no para de sacar nuevas versiones
- El código Javascript generado es cross-browser, es decir, se generan capas javascript para cada navegador.
- Comportamiento extendido a través de Smart-GWT o GWT-EXT.
- Internacionalización sencilla

- Funcionamiento contemplado del botón Atrás (Back) a pesar de contar con ciclos de vida de Ajax

#### Contras

- Problemas de compatibilidad entre las distintas versiones disponibles
- Pocos libros de las últimas versiones
- La traducción de código Java a Javascript es muy lenta



## 4. SOLUCIÓN APORTADA

### 4.1 Introducción

CRUD-Framework es un marco de trabajo que permite facilitar el desarrollo de aplicaciones empresariales. Ofrece una arquitectura de desarrollo, así como una herramienta de generación automática de CRUDs que mejora la productividad de los equipos de desarrollo, lo que lo convierte en una técnica RAD (*Rapid Application Development*).

### 4.2 Concepto y definición

De esta forma, se pueden desarrollar aplicaciones en un par de horas, incluso en unos minutos, listas para ser desplegadas. Puesto que la mayoría de aplicaciones de gestión suelen tener CRUDs de entidades, como punto de partida en el desarrollo de aplicaciones, todas estas funcionalidades estarían cubiertas. Evidentemente, los requisitos de un proyecto difícilmente son tan simples, por lo que se ofrecen mecanismos extensibles que nos permitirán modificar el comportamiento por defecto.

Otro gran beneficio que ofrece CRUD-Framework es mejorar el tiempo de adaptación o aprendizaje de los miembros del equipo de desarrollo a las tecnologías utilizadas en dicho *framework* ya que al generar todo el código de una entidad se puede observar fácilmente todas las creaciones de instancias en cada uno de las capas:

- **DAO** (*data access object*): Son los objetos de acceso a datos que contienen todas las operaciones de acceso a base de datos: consultas, inserciones, actualizaciones, borrados, etc.
- **SERVICE, MANAGER o BO** (*business object*): Son los servicios de lógica de negocio relativos al dominio del proyecto en cuestión. Es en este nivel donde se debe encontrar el núcleo de funcionalidad que aunque una serie de servicios genéricos
- **MANAGED BEANS**: Son los *beans* usados por JSF en la capa de vista para asociar los datos de las páginas JSF. Para cada entidad se crearán dos: uno para detalle y otro para listado.
- **NAVIGATION RULES** (reglas de navegación): JSF define el control de saltos de una página a otra a través de reglas de navegación.
- **VISTAS**: Son las páginas XHTML usadas en la capa de presentación. Se crearán dos por entidad: `form.xhtml` (detalle) y `list.xhtml` (listado).

Se demostrará como este framework corresponde a una técnica RAD al generar una aplicación en unas horas y únicamente dedicar algún tiempo a extender el comportamiento del CRUD en base a las peculiaridades de los casos de uso definidos a partir de las especificaciones de los requisitos dados por el cliente.

### 4.3 Generación automática de CRUDs a partir de entidades JPA

Crudfw además de ofrecer una arquitectura de referencia para diseñar aplicaciones de gestión siguiendo el patrón CRUD para cada entidad, también ofrece un mecanismo de generación automática de código en forma de plugin Maven:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>crudgenerator-maven-plugin</artifactId>
  <version>1.0-SNAPSHOT</version>
  <configuration>
    <genericCore>false</genericCore>
    <fullSource>false</fullSource>
    <componentProperties>
      <commonPackage>es.uma.crudframework</commonPackage>
      <authorName>${developer.company}</authorName>
      <configurationfile>
        src/main/resources/hibernate.cfg.xml
      </configurationfile>
      <sampleDataFile>
        src/test/resources/sample-data.xml
      </sampleDataFile>
      <menuPage>
        /src/main/webapp/common/menu/horizontal.xhtml
      </menuPage>
    </componentProperties>
  </configuration>
  <dependencies>
    <dependency>
      <groupId>${jdbc.groupId}</groupId>
      <artifactId>${jdbc.artifactId}</artifactId>
      <version>${jdbc.version}</version>
    </dependency>
  </dependencies>
</plugin>
```

### 4.4 Arquitectura

CrudFramework se compone de dos elementos estructurales totalmente distintos. Por un lado de un plugin Maven y por otro de una serie de módulos comunes que faciliten la generación de aplicaciones.

El plugin Maven corresponde al artefacto siguiente que se encuentra en `/crudfw/crudfw-plugins/crudgenerator-maven-plugin`:

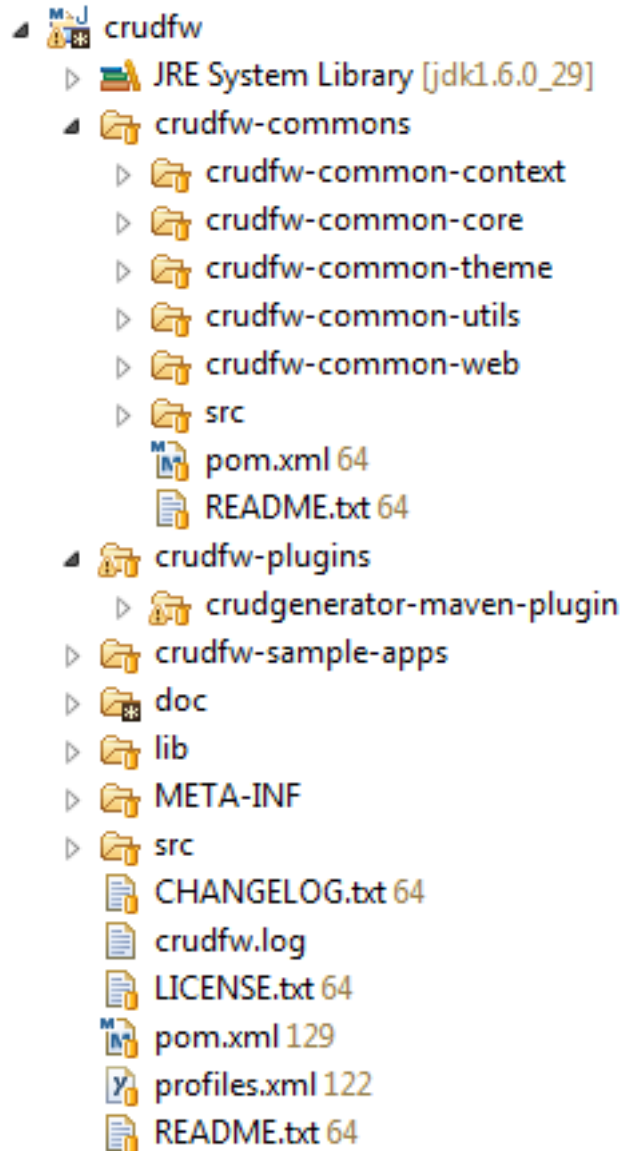
```
artifactId: crudgenerator-maven-plugin
groupId: org.codehaus.mojo
version: 1.0-SNAPSHOT
```

Dicho *plugin* permite ser invocado desde un proyecto Maven y ofrece dos operaciones:

- **gen-core**: Genera toda la capa **core** de una entidad. Genera toda la funcionalidad de una entidad: acceso a datos, servicios de lógica de negocio, etc.

- **gen-web:** Genera toda la capa **web** de una entidad. Genera la capa de presentación de una entidad. Esto es, su contenido web, sus clases de control y todo lo relacionado con la capa de vista.

A continuación se muestra todos los artefactos de CrudFramework:



Módulos comunes:

- **crudfw-common-core:** Módulo común con clases genéricas y abstractas ofreciendo la lógica de negocio de un CRUD entre las que destacan las clases como el `GenericManager` y las operaciones de acceso a datos, usando Hibernate, en el `GenericDAO`.
  - El `GenericManager<T, PK>` (siendo `T` el tipo de la entidad y `PK` el tipo del identificador o **Id**) ofrece las siguientes operaciones:
    - `Long countAll():` Devuelve el número de elementos de cierta entidad
    - `List<T> getAll():` Devuelve todos los elementos de una entidad

- `List<T> getPage(int pag, int numRes)`: Devuelve un listado de elementos paginado indicando la página y el número de resultados por página.
  - `List<T> getPage(int pag, int numRes, String orderColumn, boolean ascendingOrder)`: Ídem que la función anterior, campo de ordenación y el sentido del order.
  - `T get(PK id)`: Obtiene un elemento a partir del identificador de clave primaria.
  - `boolean exists(PK id)`: Ídem que el método anterior pero devuelve `true` si existe y `false` en otro caso.
  - `T save(T object)`: Persiste una instancia de una entidad en la capa DAO y devuelve la entidad guardada.
  - `void remove(PK id)`: Elimina una entidad en la capa DAO partir de su identificador.
- El `GenericDAO<T, PK>` (siendo `T` el tipo de la entidad y `PK` el tipo de la clave primaria o **Id**) ofrece las siguientes operaciones:
  - `T createEmptyEntity()`: Crea una entidad nueva.
  - `Long countAll()`: Devuelve el número de elementos de cierta entidad.
  - `List<T> getAll()`: Devuelve todos los elementos de una entidad
  - `List<T> getAllDistinct()`: Devuelve todos los elementos de una entidad sin duplicados.
  - `List<T> findByNameQuery(String queryName, Map<String, Object> queryParams)`: Devuelve una lista de entidades a partir de una *named query*.
  - `List<T> getPage(int pag, int numRes)`: Devuelve un listado de elementos paginado indicando la página y el número de resultados por página..
  - `List<T> getPage(int pag, int numRes, String orderColumn, boolean ascendingOrder)`: Ídem que la función anterior, campo de ordenación y el sentido del order.
  - `T get(PK id)`: Obtiene un elemento a partir del identificador de clave primaria
  - `boolean exists(PK id)`: Ídem que el método anterior pero devuelve `true` si existe y `false` en otro caso.
  - `T save(T object)`: Persiste una instancia de una entidad en base de datos y devuelve la entidad guardada.
  - `void remove(PK id)`: Elimina una entidad en base de datos a partir de su clave primaria.
  - `List<T> execSQL(String sql)`: Devuelve una lista de elementos a partir de una consulta SQL.
- **crudfw-common-web**: Módulo común con clases genéricas y abstractas ofreciendo la lógica de presentación en el contexto de JSF. Destaca la clase `BasePage` de la que heredaran todos los *managed beans* generados por el plugin Maven:
  - `BasePage`:
    - `String getParameter(String name)`: Devuelve el valor de un parámetro de la petición Http
    - `Object getSessionParam(String name)`: Devuelve un valor de un parámetro de sesión Http
    - `void setSessionParam(String name, Object value)`: Guarda un parámetro en sesión

- **crudfw-common-context:** Módulo común con la configuración del contexto de Spring configurable.
- **crudfw-common-theme:** Módulo *war overlay* usado para definir el *web content* genérico del crudfw. Contiene todo el contenido de índole web: xhtmls, facelets, imágenes, hojas de estilo y scripts *javascript*.
- **crudfw-common-utils:** Módulo que contiene clases comunes de utilidades

#### ***4.5 Fases de desarrollo***

Para desarrollar este proyecto se ha seguido un método iterativo incremental en el que se han seguido las siguientes fases:

Fase 1: Definición arquitectura de desarrollo y estructura inicial

Fase 2: Creación de la maqueta de prueba de todas las capas

Fase 3: Generación de módulos comunes a partir de la maqueta, generando clases abstractas

## 5. MANUAL DE USUARIO

En el siguiente capítulo se muestra un caso práctico de uso del framework desarrollado. Para ello se va a desarrollar una aplicación de gestión para clubes de tenis, denominada **tennisclub-manager**.

Dicha aplicación deberá ofrecer las necesidades de gestión de un club de tenis. Para ello deberá contar con los siguientes CRUDs para las siguientes entidades:

- **Socios:** Un club deportivo se compone de socios. Corresponden a los accionistas de la sociedad. Tendrán un número de socio y datos de identificación personal.
- **Facturas:** Cada socio podrá solicitar facturas de las operaciones que realice en el club: Alquileres, clases impartidas por un monitor, compras de artículos deportivos, etc.
- **Empleados:** En club de tenis se identifican varios tipos de empleados: Monitores, oficinistas, jardineros, limpiadores, etc. Tendrán un código de empleado
- **Torneos:** Un club de tenis puede organizar torneos en el que se inscriben jugadores (socios o no). Cada torneo se compone de una serie de cuadros. Cada cuadro se compone de un conjunto de partidos (disputados o pendientes de disputar).
- **Cuadros:** Cada torneo tiene diversos cuadros en función de la categoría: Absoluto, cadete, infantil, benjamín, etc.
- **Partidos:** Cada partido deberá registrar los jugadores enfrentados y el resultado, así como algún tipo de incidencia o comentario.
- **Jugadores:** Serán los jugadores inscritos en un torneo. Podrán ser socios o no. Al dar de alta del jugador si se especifica el número de socio del jugador, se deberán pre-cargar el resto de datos del socio.

### 1. Dada la entidad Jugador

```
package org.tennisclub.manager.model;

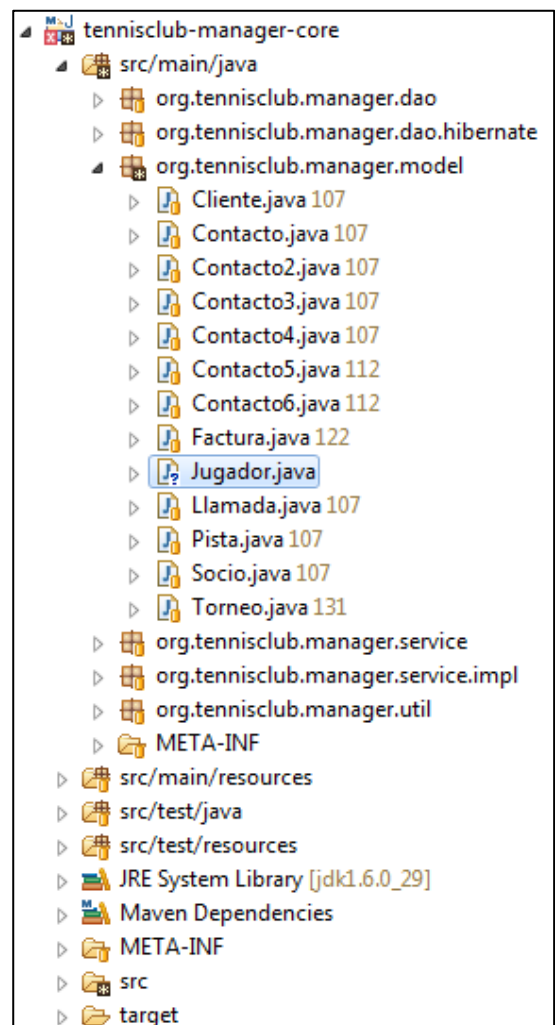
import java.util.Date;
import javax.persistence.Entity;
import es.uma.crudframework.model.BaseObject;

@Entity
public class Jugador extends BaseObject {

    // BaseObject implements Serializable
    private static final long serialVersionUID
= 1L;

    // Fields
    private String nombre;
    private String apellidos;
    private Integer numRanking;
    private Date fechaNacimiento;
    private Integer numLicencia;

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellidos() {
```



```

        return apellidos;
    }
    public void setApellidos(String value) {
        this.apellidos = value;
    }
    public Integer getRanking() {
        return ranking;
    }
    public void setRanking(Integer ranking) {
        this.ranking = ranking;
    }
    public Date getFechaNac () {
        return fechaNac;
    }
    public void setFechaNac (Date fechaNac) {
        this.fechaNac = fechaNac;
    }
    public Integer getNumLicencia() {
        return numLicencia;
    }
    public void setLicencia(Integer value) {
        this.licencia = value;
    }
}

```

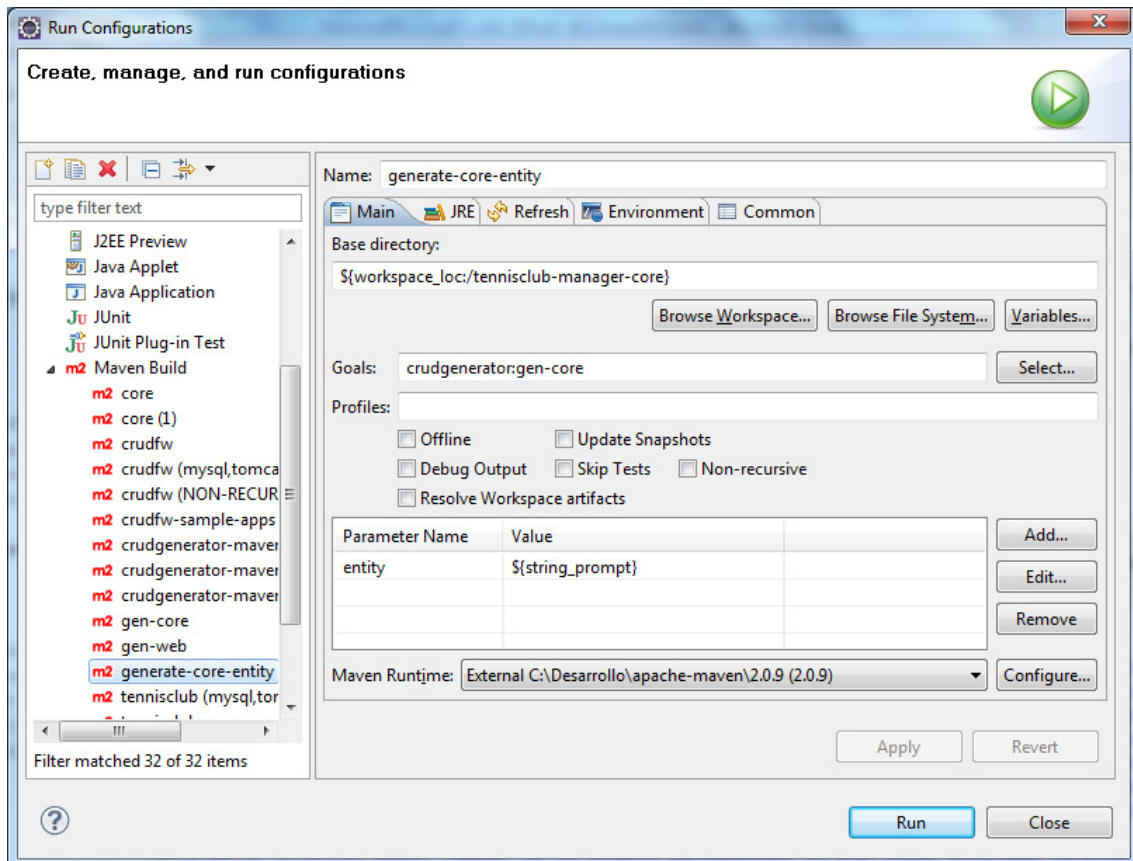
**2.** Generamos la capa **core** asociada a dicha entidad invocando al *plugin* mediante Maven. Tenemos dos formas de realizar esta operación:

a) directamente a través de línea comandos:

```
$tennis-manager/core> mvn crudgenerator:gen-core -Dentity=Jugador
```

b) o bien, a través de una *runtime configuration* en Eclipse del plugin para Maven m2eclipse:

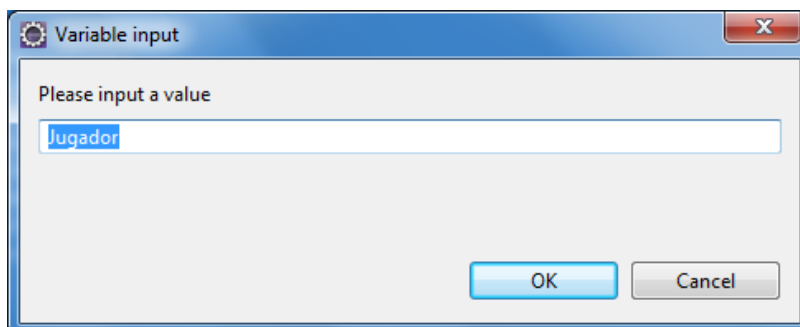
Definimos en el menú Eclipse > Run > Run configurations... > Maven build > New:



- Especificamos como *base directory* el proyecto a partir de la opción *browse workspace...*
- Indicamos que el *goal* a ejecutar de Maven sea: `crudgenerator:gen-core`
- Por último, hacemos que al invocarse se cargue en el parámetro `entity` el valor que especifiquemos dinámicamente a partir de la variable `string_prompt`.

Para ejecutar dicha *runtime configuration*, pulsamos la opción de menú *Run > Runtime configurations.. > Maven Build > generate-core-entity*.

Lo que nos mostrará el siguiente diálogo en el que indicar el nombre de la entidad:

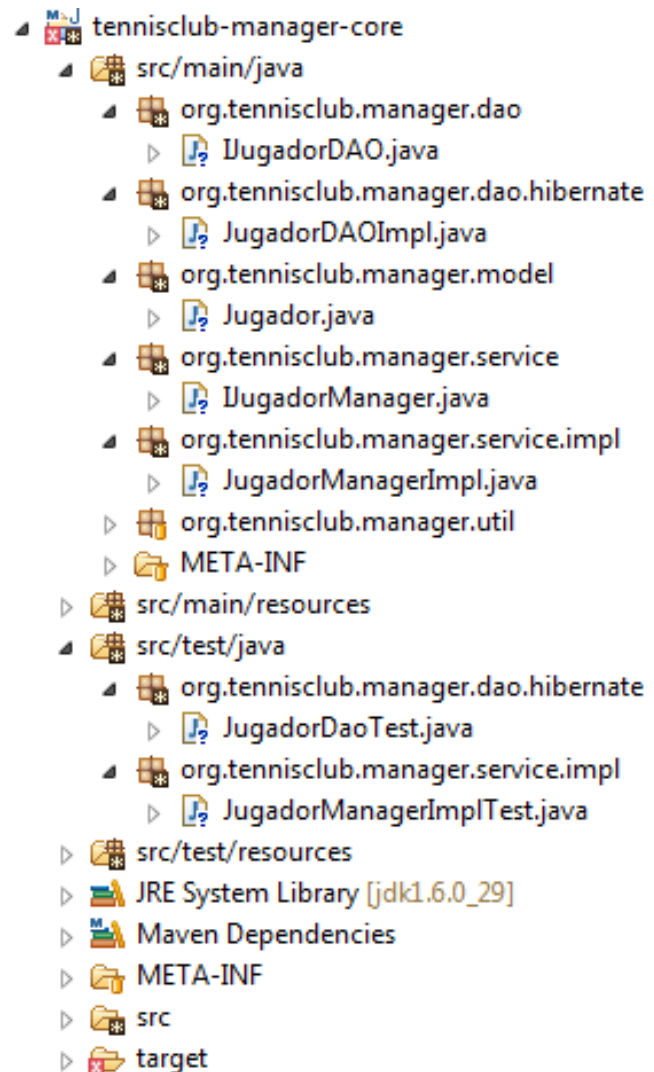


Le establecemos el nombre de nuestra entidad y una vez finalizado podemos ver el resultado obtenido:

Veamos las **clases Java generadas** a partir de la entidad `Jugador.java`:



- **Interfaz DAO:** Intefaz DAO heredero del interfaz genérico de *crudfw*: *GenericDAO*. Por tanto dicho interfaz contiene todas las operaciones de acceso a base de datos para el caso de la entidad Jugador y podrá extenderse si el caso de uso lo requiere.
- **Implementación DAO:** Implementación del DAO preparado para interactuar con el sistema de gestión de persistencia Hibernate. Es heredero de la clase de *crudfw*: *GenericDAOImpl*. Por tanto contiene como punto de partida la funcionalidad del DAO genérico de Hibernate, pudiendo este ser ampliado o modificar el comportamiento por defecto.
- **Interfaz Manager:** Intefaz Manager heredero del interfaz genérico de *crudfw*: *GenericManager*. Por tanto dicho interfaz contiene todas las operaciones de un *manager* o BO con lógica de negocio de CRUD. En este caso es muy probable que necesitemos extender su funcionalidad, pero como punto de partida es un CRUD.
- **Implementación Manager:** Implementación del Manager preparado ofrecer una lógica de negocio de un CRUD de una entidad y ser extendido por los requisitos de cada caso de uso.
- **Test DAO:** Test unitario implementado en JUnit para verificar todas las operaciones del DAO de la entidad generada (Más adelante veremos cómo se efectúa la carga inicial en base de datos de dicha entidad)
- **Test Manager:** Test de integridad o Mock que sirve para verificar la integridad del manager en su relación con el resto de objetos con los que debe interactuar.



Veamos ahora los ficheros **de recursos de configuración** modificados automáticamente para la entidad generada:

- **Instancia Spring para el DAO:** Se crea la instancia del DAO en el contexto de Spring para poder ser instanciada por la capa Manager añadiendo el siguiente *bean* al fichero `/src/main/resources/spring/applicationContext-dao.xml`.

```
<bean id="jugadorDAO"
      class="org.tenniscub.manager.dao.hibernate.
              JugadorDAOImpl">
    <property name="sessionFactory"
              ref="sessionFactory"/>
</bean>
```

- **Instancia Spring para el Manager:** Se crea la instancia del Manager en el contexto de Spring para poder ser instanciada por el *managed bean* de JSF añadiendo el siguiente *bean* al fichero `/src/main/resources/spring/applicationContext-service.xml`:

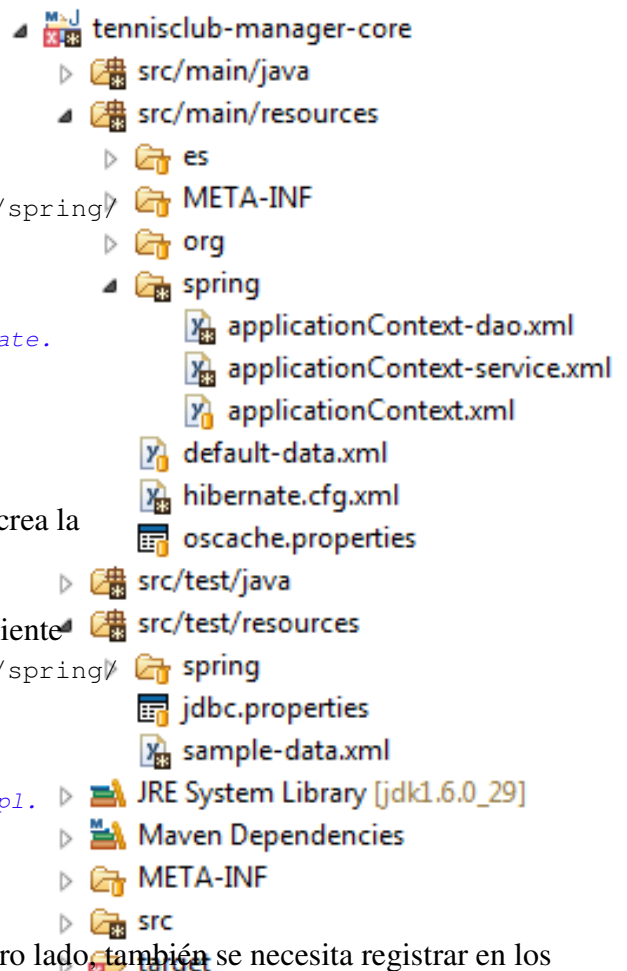
```
<bean id="jugadorManager"
      class="org.tenniscub.manager.service.impl.
              JugadorManagerImpl">
    <constructor-arg ref="jugadorDAO"/>
</bean>
```

- **Mapeo de entidad en Hibernate:** Por otro lado, también se necesita registrar en los *listeners* de Hibernate que la entidad Jugador debe ser gestionada por Hibernate. Para ello, se debe añadir la siguiente línea al fichero `/src/main/resources/hibernate.cfg.xml`:

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate
Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <mapping class="org.tenniscub.manager.model.Jugador"/>
    </session-factory>
</hibernate-configuration>
```

- **Datos de carga iniciales para tests:** Para poder ejecutar satisfactoriamente los tests del DAO de la entidad Jugador, es necesario contar con datos de prueba. Para ello, se añadirán al fichero `/src/test/resources/sample-data.xml`, lo siguiente:

```
<?xml version='1.0' encoding='UTF-8'?>
<dataset>
    <!--Jugador-START-->
    <table name="Jugador">
        <column>id</column>
        <column>apellidos</column>
```



```

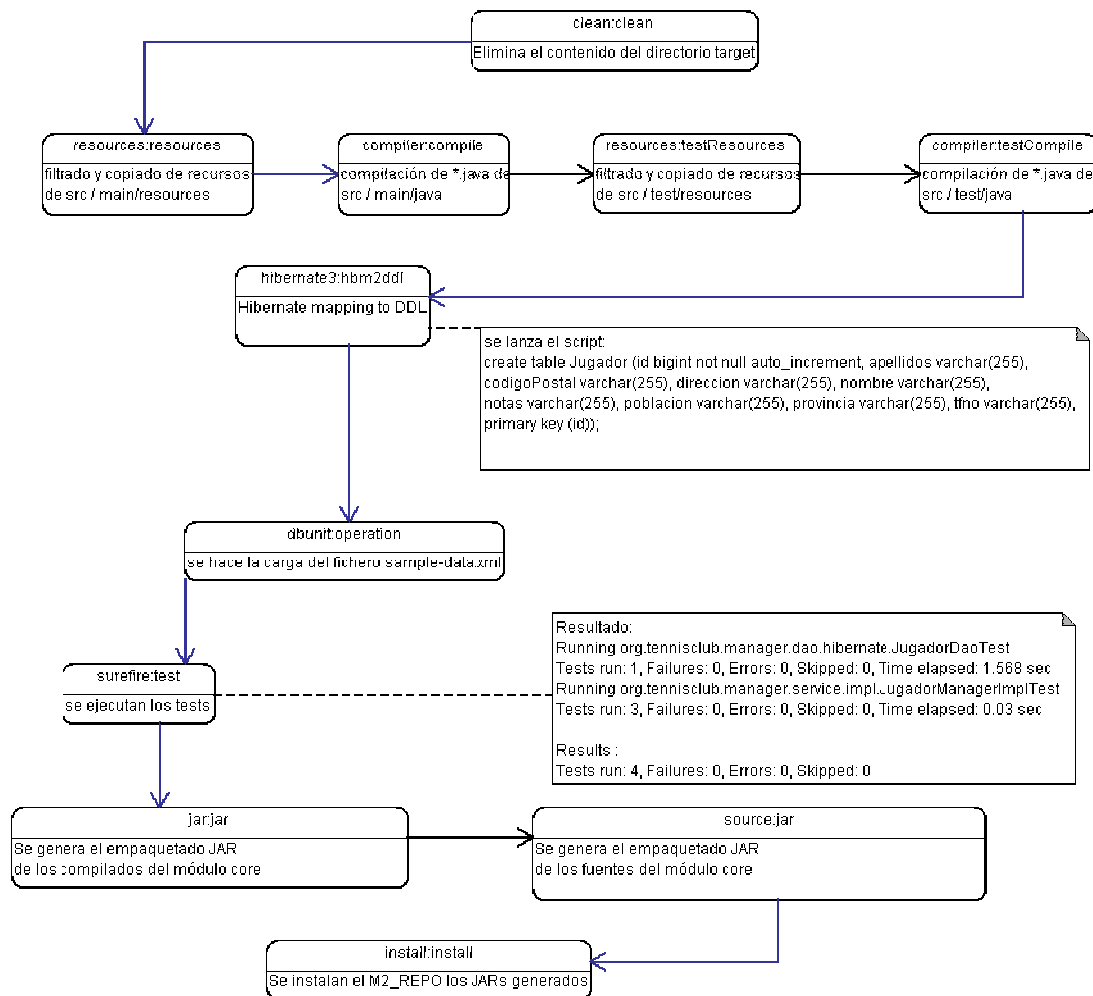
        <column>codigoPostal</column>
        <column>direccion</column>
        <column>nombre</column>
        <column>notas</column>
        <column>poblacion</column>
        <column>provincia</column>
        <column>tfno</column>
    <row>
        <value description="id">-1</value>
        <value description="apellidos">Bo..sS</value>
        <value description="codigoPostal">Cw..xV</value>
        <value description="direccion">Rj..iI</value>
        <value description="nombre">Lm..aZ</value>
        <value description="notas">Ed..iF</value>
        <value description="poblacion">Ls..tO</value>
        <value description="provincia">Ob..qL</value>
        <value description="tfno">Ab..lS</value>
    </row>
    <row>
        <value description="id">-2</value>
        <value description="apellidos">Nd..zE</value>
        <value description="codigoPostal">Fc..uE</value>
        <value description="direccion">Hz..gP</value>
        <value description="nombre">He..jB</value>
        <value description="notas">Ea..rW</value>
        <value description="poblacion">Xz..eY</value>
        <value description="provincia">Uk..fL</value>
        <value description="tfno">Si..eY</value>
    </row>
    <row>
        <value description="id">-3</value>
        <value description="apellidos">Br..pI</value>
        <value description="codigoPostal">Jz..uG</value>
        <value description="direccion">Ql..dX</value>
        <value description="nombre">Oz..fX</value>
        <value description="notas">Ty..mE</value>
        <value description="poblacion">Xr..xW</value>
        <value description="provincia">Sd..vV</value>
        <value description="tfno">Me..hD</value>
    </row>
</table>
<!--Jugador-END-->
</dataset>

```

**3. Compilamos el módulo **core**, creamos la tabla en base de datos a dicha entidad, hacemos la carga de datos inicial y ejecutamos los tests:**

```
$tennis-manager/core> mvn clean install
```

El ciclo de vida en Maven del POM del módulo core está preparado para hacer siguientes operaciones durante la ejecución de los *maven goals*: clean install:



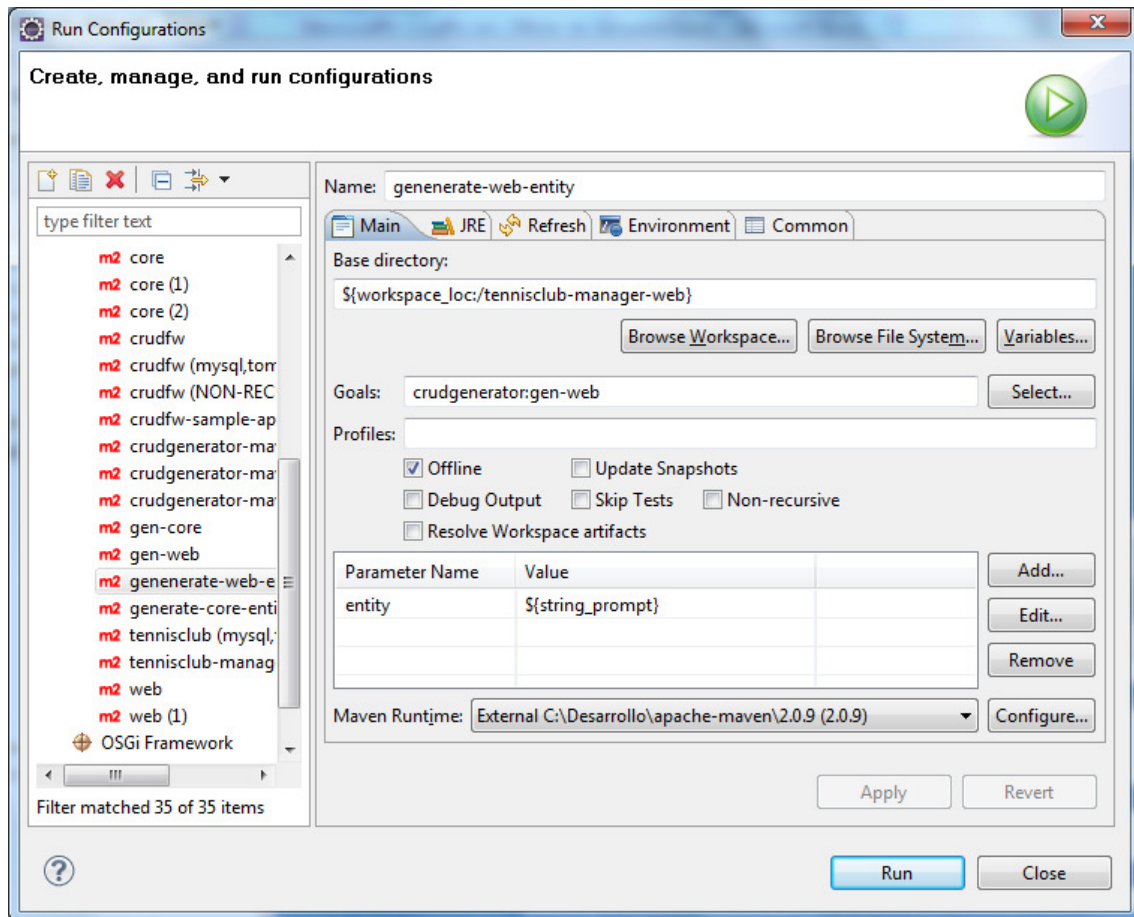
**4.** Llegado a este punto, generamos el código necesario para la capa de presentación de la entidad Jugador. Para ello invocamos al plugin Maven a través de dos opciones posibles:

a) directamente a través de línea comandos:

```
$tennis-manager/core> mvn crudgenerator:gen-web -Dentity=Jugador
```

b) o bien, a través de una *runtime configuration* en Eclipse del plugin para Maven m2eclipse:

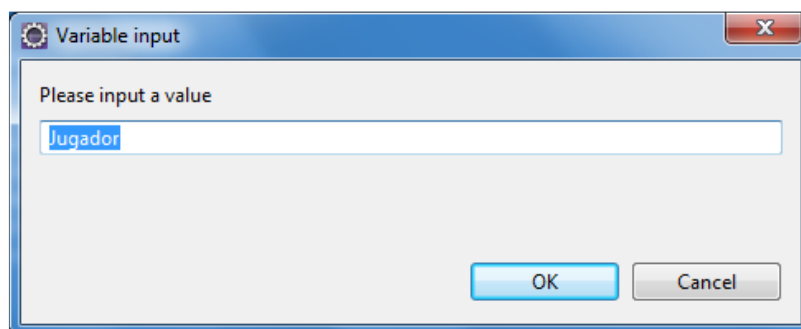
Definimos en el menú Eclipse > Run > Run configurations... > Maven build > New:



- Especificamos como *base directory* del proyecto a partir de la opción *browse workspace...*
- Indicamos que el *goal* a ejecutar de Maven sea: `crudgenerator:gen-web`
- Por último, hacemos que al invocarse se cargue en el parámetro `entity` el valor que especifiquemos dinámicamente a partir de la variable *string\_prompt*.

Para ejecutar dicha *runtime configuration*, pulsamos la opción de menú *Run > Runtime configurations.. > Maven Build > generate-web-entity*.

Lo que nos mostrará el siguiente diálogo en el que indicar el nombre de la entidad:



Le establecemos el nombre de nuestra entidad y una vez finalizado podemos ver el resultado obtenido:

Veamos las **clases Java** generadas y los **recursos de configuración** modificados automáticamente a partir de la capa **core** de la entidad Jugador:

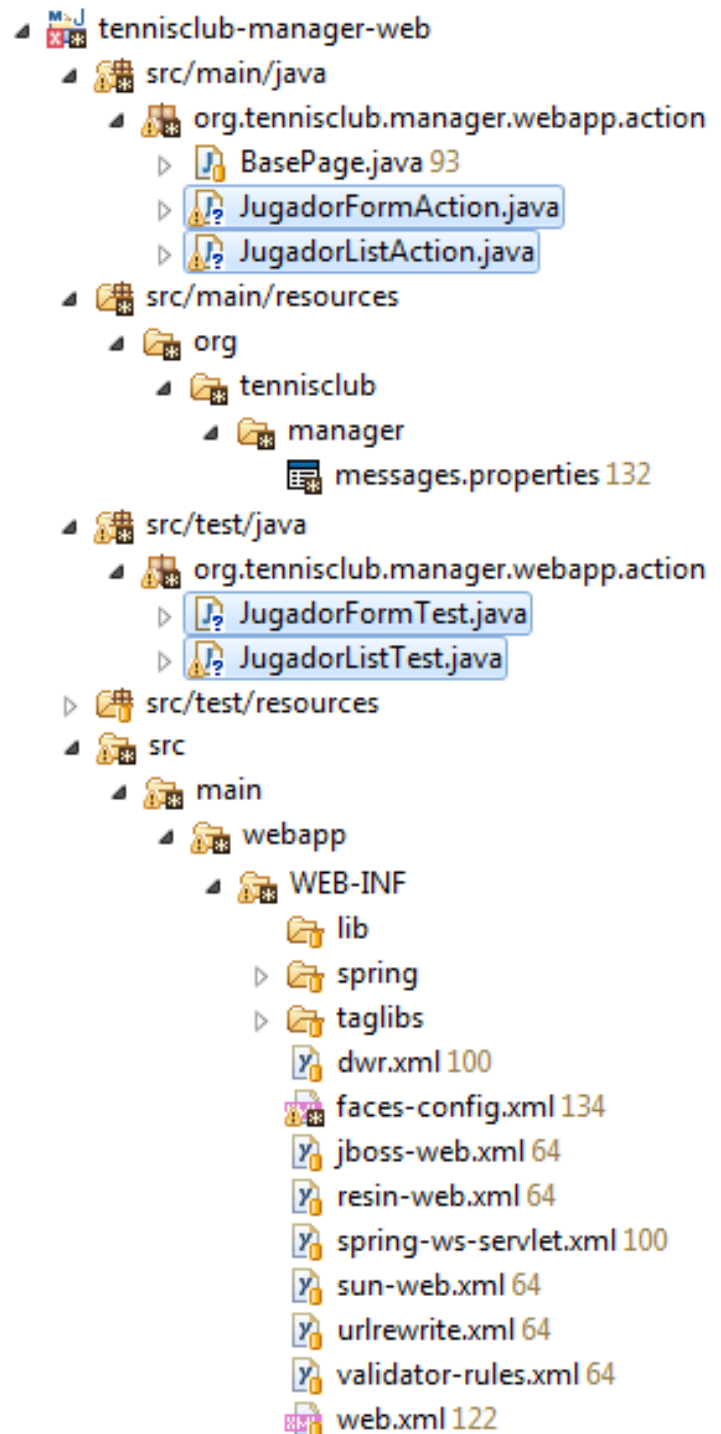
- **Managed beans** tanto para la página del listado de jugadores como para la pantalla de edición/detalle de un jugador : `JugadorListAction.java` y `JugadorFormAction.java`. Ambas clases heredan de `BasePage` que está definida en `crudfw`.
- **Tests de managed beans**. Uno por cada managed bean. Son Mocks que permiten probar la integración con la capa manager definida en el módulo **core**: `JugadorFormTest.java` y `JugadorListTest.java`.
- **Textos multidioma**: Se crean por defecto los textos de cada una de las vistas de la entidad. esto facilitará la edición de estos para adaptarlo mejor al caso de uso en cuestión. Para ello se modifica el fichero `/org/tennisclub/manager/messages.properties` añadiendo las siguientes entradas:

```
jugador.id=Id
jugador.apellidos=Apellidos
jugador.codigoPostal=Codigo Postal
jugador.direccion=Direccion
jugador.nombre=Nombre
jugador.notas=Notas
jugador.poblacion=Poblacion
jugador.provincia=Provincia
jugador.tfno=Tfno

jugador.added=Jugador has been added successfully.
jugador.updated=Jugador has been updated successfully.
jugador.deleted=Jugador has been deleted successfully.

# -- jugador list page --
jugadorList.title=Jugador List
jugadorList.heading=Jugadors
jugadorList.jugador=jugador
jugadorList.jugadors=jugadors

# -- jugador detail page --
jugadorDetail.title=Jugador Detail
jugadorDetail.heading=Jugador Information
```



- **Declaración de *Managed beans* y *navigation rules*:** Se modificará el fichero `/src/main/webapp/WEB-INF/faces-config.xml` añadiendo dos *managed beans* por cada entidad:

```
<managed-bean>
  <managed-bean-name>jugadorListAction</managed-bean-name>
  <managed-bean-class>
    org.tennistclub.manager.webapp.action.JugadorListAction
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>jugadorManager</property-name>
    <value>#{jugadorManager}</value>
  </managed-property>
</managed-bean>

<managed-bean>
  <managed-bean-name>jugadorFormAction</managed-bean-name>
  <managed-bean-class>
    org.tennistclub.manager.webapp.action.JugadorFormAction
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>id</property-name>
    <value>#{param.id}</value>
  </managed-property>
  <managed-property>
    <property-name>jugadorManager</property-name>
    <value>#{jugadorManager}</value>
  </managed-property>
</managed-bean>
```

Adicionalmente también se crearan las reglas de navegación (*navigation rules*) necesarias para definir los saltos de una pantalla a otra:

```
<navigation-rule>
  <from-view-id>/pages/jugador/list.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>add</from-outcome>
    <to-view-id>/pages/jugador/form.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-outcome>edit</from-outcome>
    <to-view-id>/pages/jugador/form.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
<navigation-rule>
  <from-view-id>/pages/jugador/form.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>cancel</from-outcome>
    <to-view-id>/pages/jugador/list.xhtml</to-view-id>
    <redirect/>
  </navigation-case>
  <navigation-case>
    <from-outcome>list</from-outcome>
    <to-view-id>/pages/jugador/list.xhtml</to-view-id>
    <redirect/>
  </navigation-case>
</navigation-rule>
```

Por último, veamos el contenido público del *web content* de nuestro proyecto que se ha generado:

- Por cada entidad se crea un **directorio de vistas** colgando de la ruta:  
`/src/main/webapp/pages/jugador`
- En dicho directorio se crearán dos páginas XHTML: `form.xhtml` y `list.xhtml`. La primera corresponde a la vista de *edición/creación/detalle* y la segunda corresponde al *listado* de dicha entidad.
- Por último se creará una nueva entrada de menú en el fichero  
`/src/main/webapp/common/menu/horizontal.xhtml`:

```
<li>
    <a
href="#{facesContext.externalContext.request
ContextPath}/pages/jugador/list.html">
        Jugador
    </a>
</li>
```

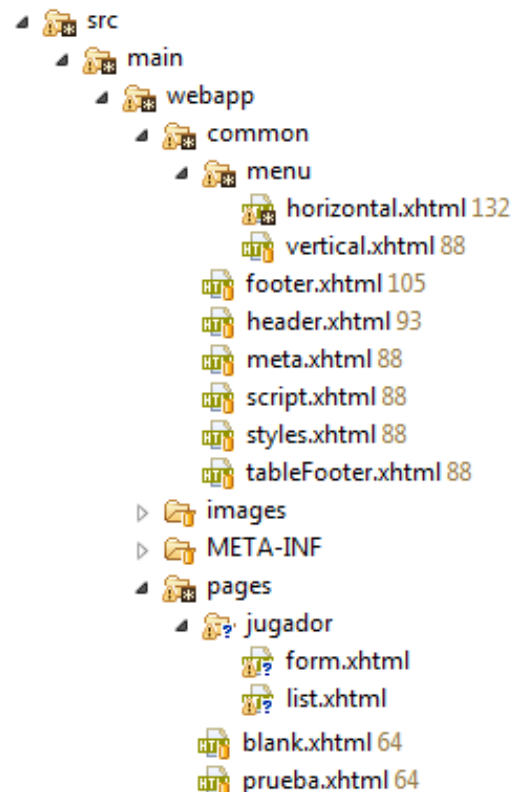
5. Por último, generamos el WAR de la aplicación a través de la orden:

```
$tennis-manager/web> mvn clean package
```


6. Si desplegamos el WAR generado `tennis-club.war` en un *Apache Tomcat 5.5* e iniciamos dicho servidor, veremos el resultado:

Si accedemos a:

<http://localhost:8070/tennis-club/pages/jugador/list.html>





 POWERED BY APACHE MAVEN2, JPA, SPRING FRAMEWORK, JSF

Home

Jugador

Inicio > Jugador List

Jugadors

Jugador List

Añadir


Id	Apellidos	Codigo Postal	Direccion	Nombre	Notas	Poblacion	Provincia	Tfno
-1	García Reyes	28080	C\11	Goya, José	Quiere cambiar el partido mañana a la misma hora	Madrid	Madrid	910000000
-2	Díaz Fernández	41190	Avda. República Argentina, 10	María		Sevilla	Sevilla	955000000
-3	Martín López	29130	C\16	Mozart, Juan	Este jugador tiene pendiente pagar la inscripción	Alhaurín de la torre	Málaga	952410000

[3 Resultados] Mostrando del 1 al 3

2010 Todos los derechos reservados Universidad de Málaga.

Política de privacidad

Y si editamos una entidad pulsando en la columna **Id** o creamos una nueva entidad pulsando en **Añadir**, veremos la siguiente pantalla:

 POWERED BY APACHE MAVEN2, JPA, SPRING FRAMEWORK, JSF

Home

Jugador

Inicio > Jugador Detail

Jugador Information

Apellidos

Díaz Fernández

Codigo Postal

41190

Direccion

Avda. República Argentina,

Nombre

María

Notas

Poblacion

Sevilla

Provincia

Sevilla

Tfno

955000000

Borrar Guardar Cancelar

2010 Todos los derechos reservados Universidad de Málaga.

Política de privacidad

## 6 CONCLUSIONES

La realización de este proyecto ha tenido como motivación la idea que siempre he perseguido de reutilizar el trabajo realizado, de tener conciencia de que un trabajo técnico como el desarrollo de software debe ser visto como un trabajo de ingeniería.

El desarrollo de software debe orientarse a técnicas que mejoren la productividad reutilizando de una forma probada y contrastada la experiencia adquirida en proyectos anteriores. Es por ello que el caso de aplicaciones de gestión de tipo CRUD pueden ser implementadas de una forma rápida y eficaz. Con ello, la dificultad de implementación de un caso de uso de una aplicación de gestión se reduciría y únicamente habría que preocuparse de las necesidades del cliente, no de la arquitectura tecnológica ni de su aprendizaje.

Paralelamente al inicio de este proyecto de fin de carrera, han aparecido diversos productos que persiguen ideas similares. Un ejemplo claro de ello es Spring Roo que ofrece servicios similares, al comportarse como una herramienta RAD que permite generar proyectos de una forma rápida y dotada de mucho apoyo de la comunidad de software libre.

Por ello, un proyecto así en su momento era muy novedoso, hoy en día corresponde al tipo de estrategias que se están siguiendo en grandes corporaciones de desarrollo software. Apostar por herramientas de desarrollo automatizadas sigue los fines que el desarrollo de aplicaciones SaaS (*service as a service*) del mundo cloud. Por ello, estamos asistiendo a una revolución tecnológica en el que la industria del software está optimizando sus procesos de desarrollo, dejando a un lado metodologías de desarrollo software obsoletas, que en su propia filosofía se incurrieran en problemas de calidad e ineficiencia en los tiempos de finalización de proyectos.

Planificar un proyecto es una ardua tarea. Si se cuenta con herramientas RAD como la desarrollada en este proyecto, pasaríamos a planificar reduciendo los riesgos al acotarlos únicamente a cada caso de uso. No obstante, siempre habrá un cierto grado de incertidumbre al enfrentarse aun proyecto nuevo.

Contar con una herramienta así no se debe ver de una forma externa, es preciso mantenerla e ir enriqueciéndola continuamente, incluyendo nuevas capacidades de desarrollo automatizado. De todos modos, debe haber un compromiso entre eficiencia, reutilización y costes. Por ello, se debe de analizar constantemente qué desarrollos se deben automatizar y cuáles no interesa.

El apartado anterior nos darnos cuenta qué la herramienta actual podría ser ampliada. Por ello, podríamos considerar las siguientes propuestas de ampliación:

- Generación automatizada de pantallas de búsqueda de entidades, usando mecanismos genéricos de filtrado y adicionalmente, un sistema de indexación como puede ser Solr en los servicios genéricos de acceso a datos del módulo **crudfw-common-core**.
- Considerar el almacenamiento en Cloud en la configuración de la capa DAO (*Data Access Service*)
- Inclusión de un *middleware* SOA (*Service Oriented Architecture*) como repositorio de servicios
- Etc.

Por último, ya sí que sí, destacar las repercusiones personales que ha tenido este proyecto en mi carrera profesional.

Ha sido la fructificación del convencimiento de que un ingeniero informático no es un mero programador. Somos programadores sí, creamos autómatas, por supuesto, pero somos algo más, podemos ser meta-programadores. Aunque partamos de los mismos elementos que un programador, es nuestra obligación mejorar los procesos productivos, creando, reutilizando y usando. Por ello, el carácter de un ingeniero debe ser visible siempre de dos formas, reduciendo costes y mejorando la calidad de nuestras creaciones.

## 7 REFERENCIAS

### 7.1. RAD:

Reconfiguring the User: Using Rapid Application Development  
<http://sss.sagepub.com/content/30/5/737.short>

Rapid Application Development: An empirical review  
<http://www.ingentaconnect.com/content/pal/0960085x/1999/00000008/00000003/3000325>

ODRA: A Next Generation Object-Oriented Environment for Rapid Database Application Development  
<http://www.springerlink.com/content/f850736882164151/>

Spring Roo in Action  
K.Simple, S.Penchikala, G.Dickens  
Manning

Model2Roo: Web Application Development based on Eclipse Modeling Framework and Spring Roo  
Juan Castrejón - University of Grenoble  
ACME 2012

### 7.2. Eclipse:

The Eclipse Foundation  
<http://www.eclipse.org/>

Eclipse Galileo Documentation  
<http://help.eclipse.org/galileo/index.jsp>

Eclipse for Dummies  
Barry Burd

### 7.3. Apache Maven:

Apache Maven  
<http://maven.apache.org/>

Developing with Eclipse and Maven  
Tim O'Brien, Jason van Zyl, Igor Fedorenko, Brian Fox, Rich Seddon, Dan Yocum  
Sonatype - Maven Integration for Eclipse

Maven - The Complete Reference  
Tim O'Brien, Jason van Zyl, Brian Fox, John Casey, Juven Xu, Thomas Locher, Manfred Moser  
Sonatype - Maven Integration for Eclipse

#### **7.4. Servidores J2EE:**

Apache Tomcat

<http://tomcat.apache.org/>

Apache Tomcat 5.5 Documentation

<http://tomcat.apache.org/tomcat-5.5-doc/index.html>

Professional Apache Tomcat 5

Vivek Chopra, Amit Bakore, Ben Galbraith, Sing Li, Chanoch Wiggers  
Wrox

#### **7.5. SGBDs:**

MySQL

<http://www.mysql.com/>

MySQL Documentation

<http://dev.mysql.com/doc/>

MySQL

Paul DuBois  
Addison-Wesley

#### **7.6. ORMs:**

Spring Persistence with Hibernate

Ahmad Seddighi  
Packt Publishing

Just Spring Data Access

Madhudsudhan Konda  
O'reilly

#### **7.7. iBATIS:**

iBatis in action

Clinton Begin, Brandon Goodin, Larry Meadors  
Manning

#### **7.8. Hibernate:**

Hibernate: A J2EE Developer's Guide

Will Iverson  
Addison-Wesley Professional

## **7.9. JPA:**

Pro EJB 3.0: Java Persistence API

Mike Keith, Merrick Schincariol

Apress

## **7.10. Spring Framework:**

Spring Framework Documentation:

<http://static.springsource.org/spring/docs/2.5.x/reference/index.html>

Spring WS Documentation:

<http://static.springsource.org/spring-ws/sites/2.0/>

Professional Java Development with the Spring Framework

Rod Johnson, Juergen Hoeller, Alef Arendsen, Thomas Risberg, Colin Sampaleanu

Wiley

## **7.11. CAS:**

Central Authentication Service (CAS):

<http://www.jasig.org/cas>

## **7.12. Spring Security:**

Spring Security:

<http://static.springsource.org/spring-security/site/>

Spring Security 3

Peter Mularien

Packt Publishing

## **7.13. Spring MVC:**

Expert Spring MVC and Web Flows

Seth Ladd, Keith Donald

Apress

## **7.14. Apache Struts 2:**

Apache Struts 2: Web application development

Dave Newton

Packt Publishing

### **7.15. JSF:**

The Complete Reference: JavaServer Faces

Chris Shalk, Ed Burns

McGraw Hill

Building Web-based User Interfaces: JavaServer Faces

Hans Bergsten

O'reilly

### **7.16. GWT:**

Google Web Toolkit Applications

Ryan Dewsbury

Prentice Hall

Google Web Toolkit Solutions

David Geary, Rob Gordon

Prentice Hall

### **7.17. Comparativas de Frameworks MVC:**

Comparing Web Frameworks

Matt Raible

Raible Designs 2007