Proyecto final de carrera Ingeniería Informática

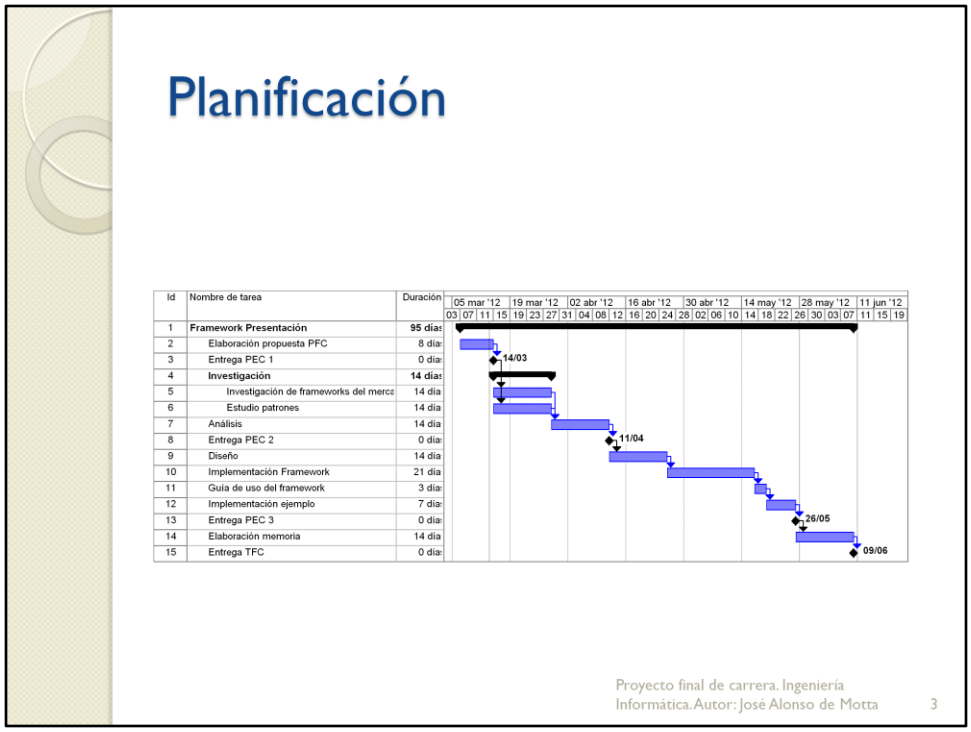
Diseño e implementación de un marco de trabajo de presentación para aplicaciones J2EE

'MTP'

Autor: José Alonso de Motta
alonsomotta@gmail.com
Consultor: Óscar Escudero Sánchez

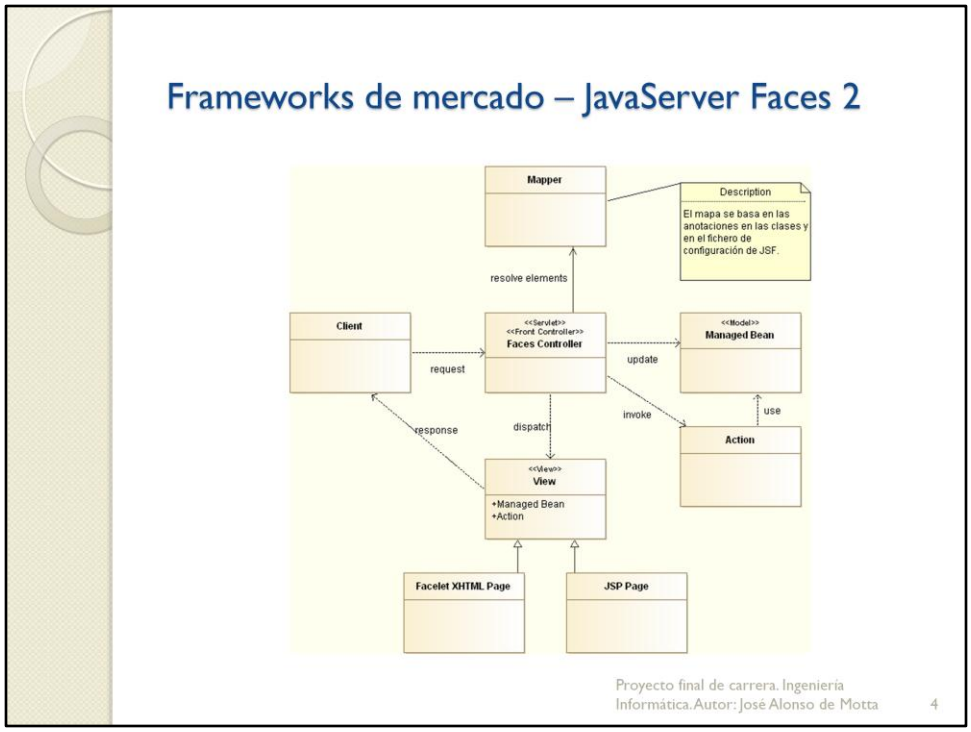
Agenda

- Planificación
- Análisis de frameworks de mercado
- Patrones
- Framework MTP
- Aplicación ejemplo AMPA
- Productos obtenidos
- Resumen



El proyecto se ha dividido en cuatro **etapas** principales:

- Se ha comenzado con la **investigación de frameworks de mercado**, donde se han analizado tres de los más representativos marcos de trabajo del área de presentación para aplicaciones J2EE.
- A la vez se ha realizado el **estudio de patrones de diseño**, donde se ha profundizado en el patrón Modelo-Vista-Controlador, un conjunto de patrones de diseño específicos para J2EE del libro «Core J2EE Patterns» y los patrones recopialdos por el cuarteto de autores conocido como «Gang of Four».
- Una vez con las bases de conocimiento necesarias, se ha procedido a realizar el **análisis del framework**, donde se han definido las características que debe tener el marco de trabajo a construir. Con las características definidas, se realiza el **diseño**, donde se definen las relaciones entre los componentes y se ha concretado el ciclo de vida de las peticiones. En la fase de **implementación** se ha materializado el framework en forma de una librería JAR que debe ser incorporada por las aplicaciones usuarias.
- Finalmente, para demostrar el uso del framework se ha construido una **aplicación de ejemplo**, de gestión de socios de una AMPA, que usa el marco de trabajo. En esta aplicación se han aplicado patrones de diseño J2EE para construir las diferentes capas de la aplicación.



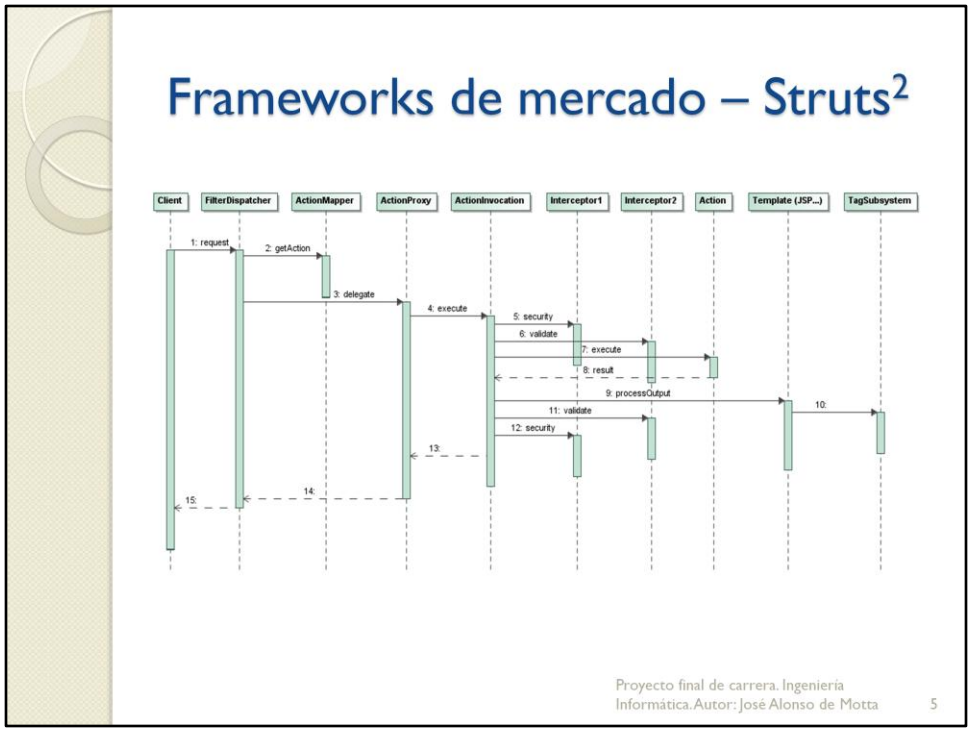
Creado por varios fabricantes para establecer un **estándar en la gestión del interfaz de usuario** de aplicaciones web para clientes ligeros con J2EE.

Aplica el **patrón Modelo-Vista-Controlador**, siendo el **controlador** un servlet (FacesServlet), los **modelos** se implementan con Managed Beans y pueden actuar como transfer objects. Las acciones (Action) que implementan los procesos de negocio pueden ser invocadas con el patrón Command. Las **vistas** se representan generalmente con Facelets XHTML o páginas JSP.

El **ciclo de vida** de una petición tiene estos pasos:

- Creación de la vista o recuperación de la vista anterior (en los procesos con varios pasos)
- Lectura de los valores de la petición que llega y aplicación de los valores a la vista creada o recuperada.
- Se realizan las validaciones necesarias.
- Se actualiza el modelo, invocando a los Managed Beans.
- Invoca a los métodos de negocio (Action) para realizar las acciones necesarias.
- Recupera la página de respuesta, la formatea de forma apropiada según el tipo de cliente.

La **configuración** en una aplicación consiste en: la declaración del Faces Servlet en el descriptor de despliegue (web.xml); personalización de la configuración en el fichero faces-config.xml; e inclusión de las librerías de JSF, generalmente en la carpeta WEB-INF/lib.



Desarrollado por Apache, esta segunda generación de Struts se basa en el producto WebWork, que fue una escisión del primer Struts.

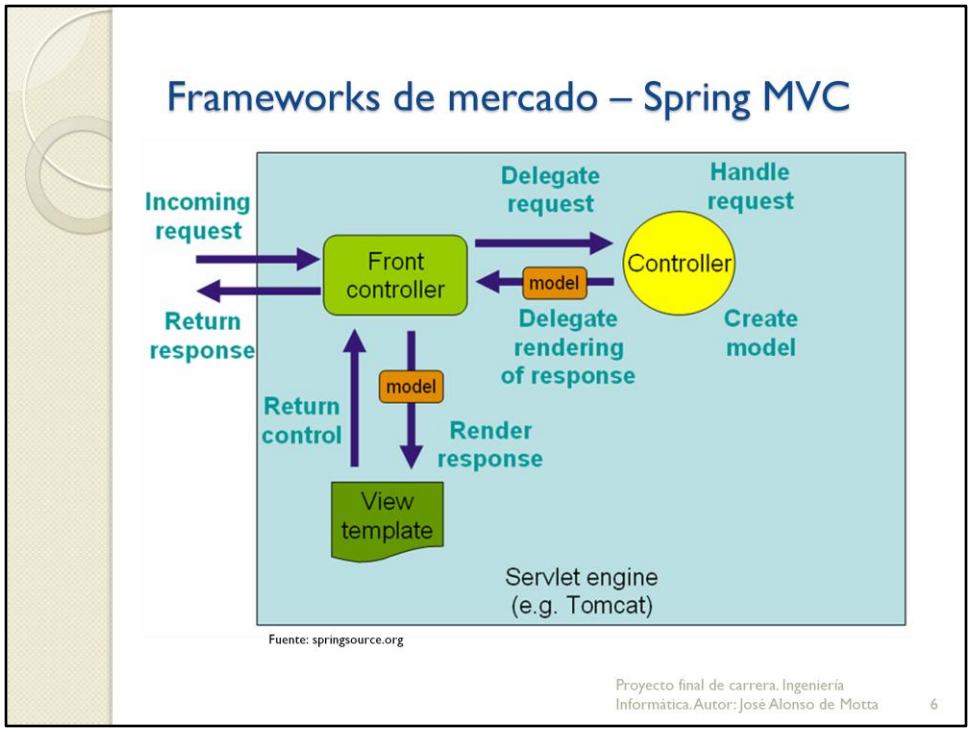
En la implementación del **patrón MVC** el **controlador** es un Servlet Filter, que procesa las peticiones y decide la acción a ejecutar. El **modelo** se implementa con el patrón Command, mediante clases Action, que ejecutan la lógica de negocio y actúan también como Transfer Object para servir como vehículo de intercambio de información entre usuario y aplicación. La respuesta del Action le sirve al framework para decidir la vista a la que redirigir la respuesta al usuario. La **vista** se implementa con páginas JSP, plantillas Velocity, XSLT, Tiles…

Ciclo de vida del request:

- El request lo recibe el FilterDispatcher (patrón Intercepting Filter), que escoge la acción a ejecutar mediante el ActionMapper (patrón Mapper).
- Para invocar a la acción utiliza el ActionProxy, que se encarga de obtenerla y en el ActionInvocation que la ejecuta, utilizando una serie de interceptores que realizan tareas genéricas como seguridad, logging, validación de parámetros y conversión a tipos Java.
- Una vez finalizada la acción, con el valor devuelto se decide la vista a cargar, la cual se puede apoyar en el subsistema de etiquetas (patrón View Helper).

La información del request permanece en todo momento accesible a través del **ValueStack**, un objeto que se almacena en el hilo de ejecución.

La **configuración** se realiza: declarando el FilterDipatcher en el descriptor de despliegue; personalizando la configuración en struts.xml y otros ficheros de configuración adicionales; incluyendo anotaciones Java en las clases; y finalmente incluyendo las librerías en el classpath de la aplicación (por ejemplo en WEB-INF/lib).



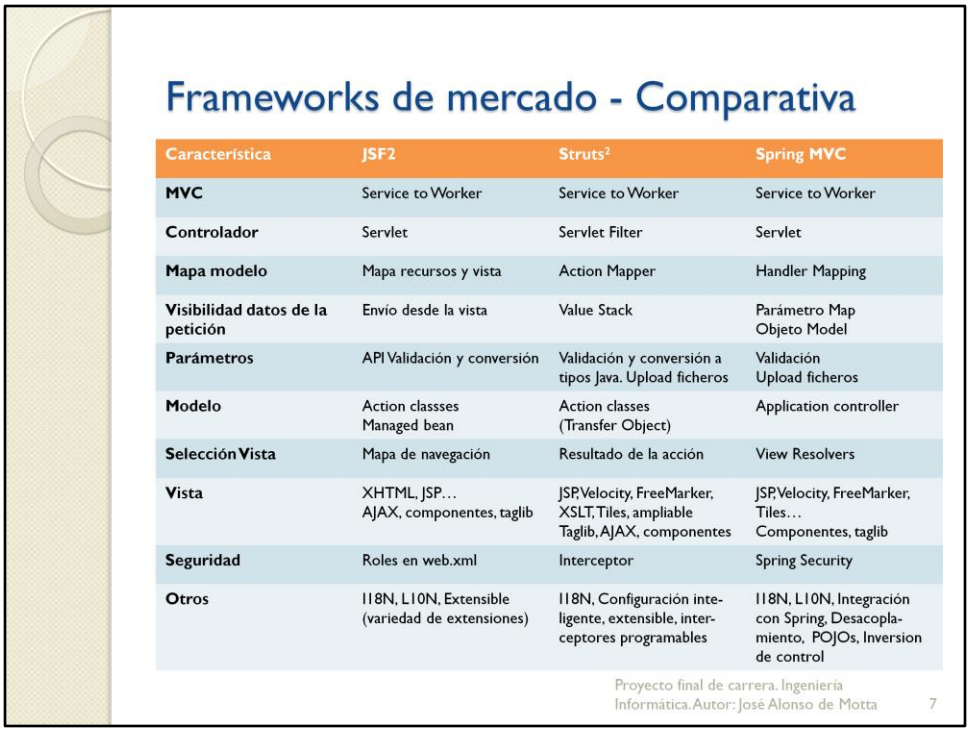
Forma **parte del framework Spring** que cubre todas las áreas de una aplicación J2EE. El desarrollador puede decidir entre integrar entre sí los distintos componentes de Spring o sustituir algunos por otros frameworks, encargándose Spring de la integración entre ellos. Las **clases Java de aplicación** no requieren la importación de interfaces o extensión de clases del framework (**POJO**). La **inversión de control** significa que es el framework el que invoca e instancia a los objetos de aplicación y no al revés.

El **controlador** principal es implementado por el DispatcherServlet. El **modelo** es representado por varios Controller (patrón Application Controller) que ejecutan el código de negocio y recuperan los datos para que el framework los entregue a la vista correspondiente. La **vista** queda representada por páginas JSP, plantillas Velocity…

El **ciclo de las peticiones** es el siguiente:

- El DispatcherServlet recibe la petición del cliente. Apoyado en uno o varios Handler Mappings (patrón Mapper) intercambiables para elegir a qué controlador de aplicación debe entregar la petición.
- El controlador de aplicación invoca a los procesos de negocio y recupera los datos para la vista, cargándolos en un parámetro recibido (de tipo Map u objeto Model) que actúa como Transfer Object. Devuelve el nombre lógico de la vista a cargar.
- Nuevamente el DispatcherServlet, apoyado en un ViewResolver (patrón Mapper) decide a qué vista invocar para devolver la respuesta al usuario.
- Se invoca a la vista, poniendo a su disposición el conjunto de datos del modelo recibido por el controlador de aplicación.

La **configuración** consiste en: declaración del servlet en el web.xml; configuración mediante fichero y/o inclusión de anotaciones en las clases implicadas; inclusión de librerías en el classpath de la aplicación (por ejemplo WEB-INF/lib).

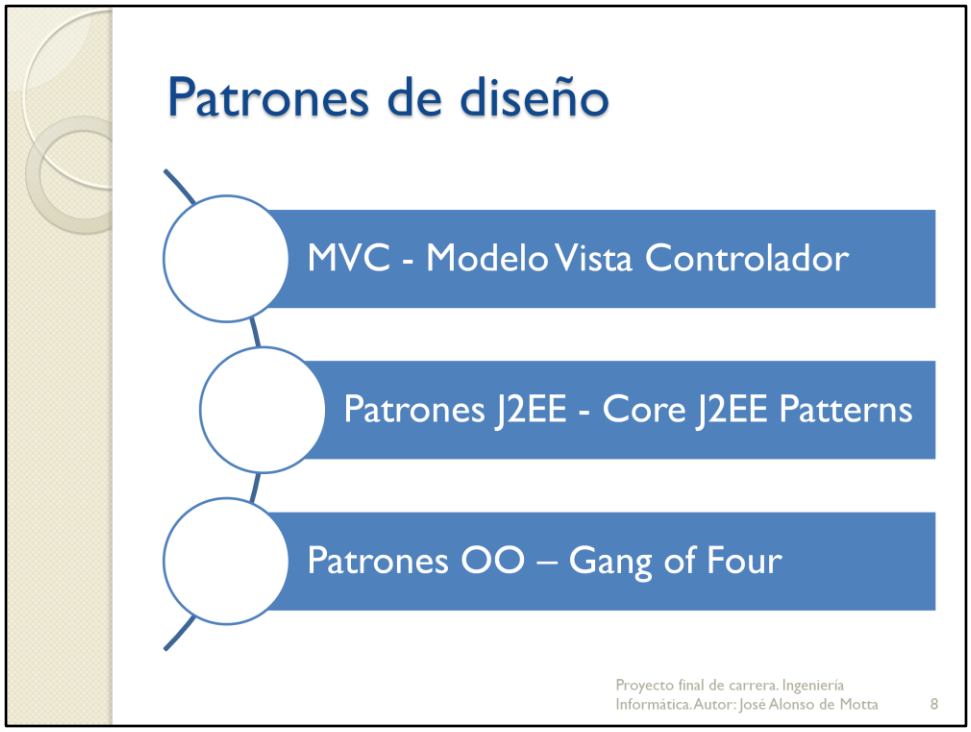


Frameworks de mercado - Comparativa

Característica	JSF2	Struts2	Spring MVC
MVC	Service to Worker	Service to Worker	Service to Worker
Controlador	Servlet	Servlet Filter	Servlet
Mapa modelo	Mapa recursos y vista	Action Mapper	Handler Mapping
Visibilidad datos de la petición	Envío desde la vista	Value Stack	Parámetro Map Objeto Model
Parámetros	API Validación y conversión	Validación y conversión a tipos Java. Upload ficheros	Validación Upload ficheros
Modelo	Action classses Managed bean	Action classes (Transfer Object)	Application controller
Selección Vista	Mapa de navegación	Resultado de la acción	View Resolvers
Vista	XHTML, JSP… AJAX, componentes, taglib	JSP, Velocity, FreeMarker, XSLT, Tiles, ampliable Taglib, AJAX, componentes	JSP, Velocity, FreeMarker, Tiles… Componentes, taglib
Seguridad	Roles en web.xml	Interceptor	Spring Security
Otros	I18N, L10N, Extensible (variedad de extensiones)	I18N, Configuración inteligente, extensible, interceptores programables	I18N, L10N, Integración con Spring, Desacoplamiento, POJOs, Inversion de control

Proyecto final de carrera. Ingeniería Informática. Autor: José Alonso de Motta 7

Se presenta un cuadro resumen con la forma de resolver diferentes problemáticas por los frameworks estudiados.



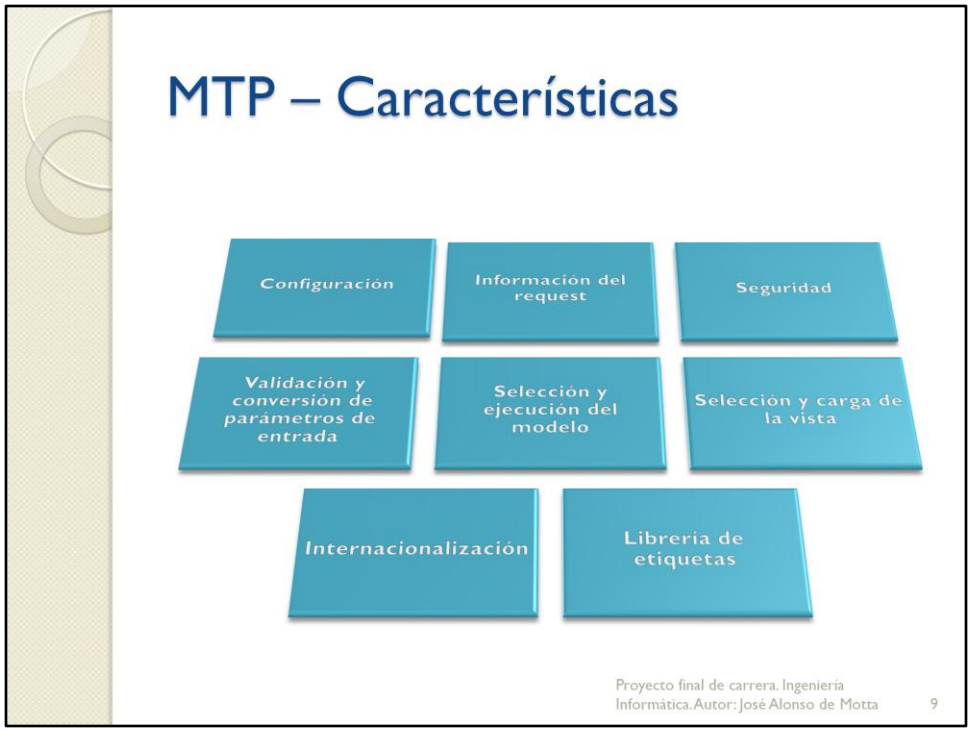
Los **patrones** son **soluciones probadas a problemáticas concretas** que aparecen durante las fases de ingeniería del software. El análisis de los patrones va a servir para comprender el funcionamiento de los marcos de trabajo examinados, siendo también fundamental para el diseño del marco de trabajo elaborado y de la aplicación de ejemplo.

En primer lugar se ha visto el **patrón Modelo-Vista-Controlador (MVC)**, eje principal de los frameworks de presentación vistos. Se basa en tres componentes:

- Controlador: Controla la lógica de la interacción usuario-aplicación.
- Modelo: Se encarga de invocar la lógica de negocio de la aplicación y de obtener los datos necesarios.
- Vista: Representa la respuesta de la aplicación ofrecida al usuario.

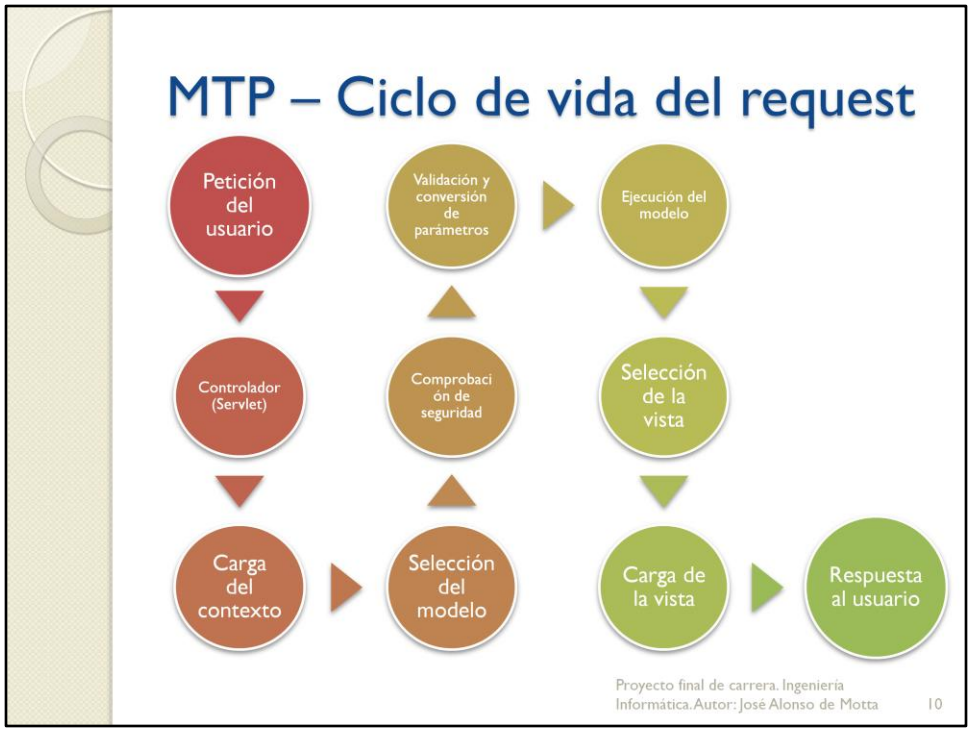
En segundo lugar se ha visto el conjunto de **patrones de diseño especializados en J2EE** del libro «Core J2EE Patterns». Este grupo de patrones ofrece soluciones para las tres capas de las aplicaciones J2EE: presentación, negocio e integración.

Finalmente se ha visto el conjunto de patrones recopilados por el cuarteto de autores conocido como **«Gang of Four»**, en el libro «Design Patterns: Elements of Reusable Object-Oriented Software». Estos patrones ofrecen soluciones a problemas comunes a los que se enfrenta la **programación orientada a objetos**.



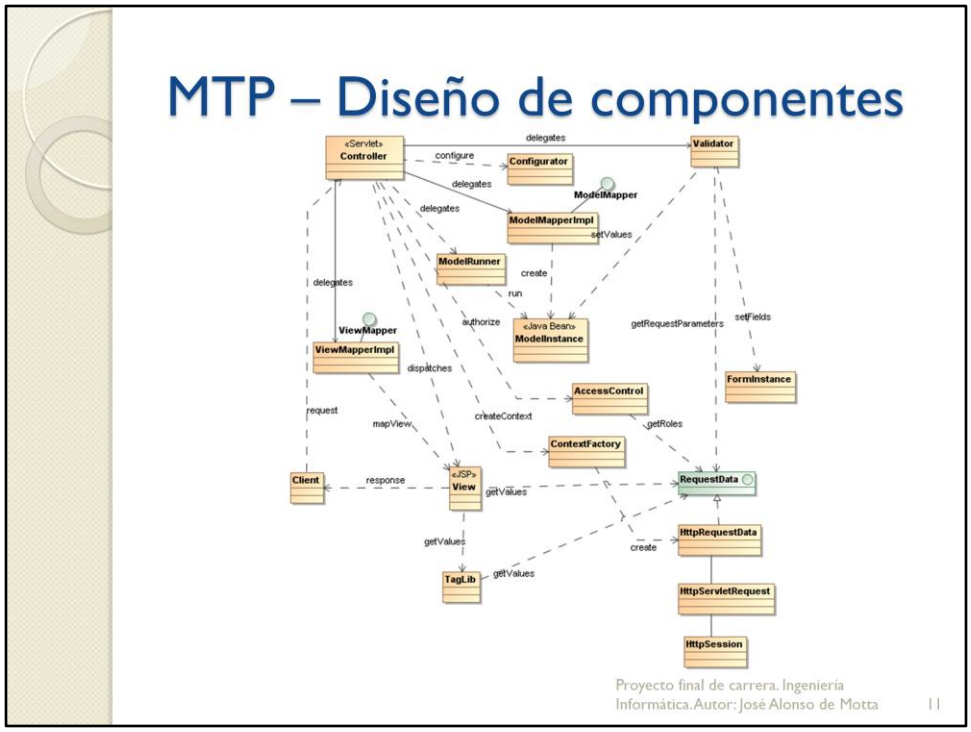
El marco de trabajo de presentación (MTP) elaborado se puede describir con sus características:

- La **configuración** se realiza en un fichero XML y en resource bundles con las traducciones del interfaz de usuario a varios idiomas.
- La **Información del request** viaja en todo momento en el objeto RequestData, que sirve también como interface para integración del framework con la aplicación.
- La **seguridad** está basada en roles, asignados por el framework a los modelos de ejecución y por la lógica de la aplicación a las sesiones de usuario.
- La **validación y conversión de parámetros de entrada** a tipos de datos Java la realiza el framework en función de la configuración realizada.
- La **selección del modelo** a ejecutar se realiza por un Mapper implementado por el framework que utiliza el parámetro MODEL del request. El marco de trabajo está abierto a que este objeto sea sustituido por uno de la aplicación que pueda implementar otro algoritmo.
- La **selección de la vista** se realiza nuevamente por otro Mapper que utiliza una correlación entre la respuesta del modelo ejecutado y una vista, punto que se define en la configuración. Este Mapper también puede ser sustituido por otro que pueda implementar la aplicación para emplear otro algoritmo.
- La **internacionalización** se soporta mediante una función del interface RequestData, una etiqueta de la librería de etiquetas y permite traducir el interface de usuario en base al idioma del usuario enviado por el navegador o a un idioma establecido para la sesión. Utiliza los resource bundles indicados en el punto de configuración.
- Se ofrece a las páginas JSP una **librería de etiquetas** que da acceso mediante tags a la información de la sesión, request y modelo y al resto de funciones del RequestData.



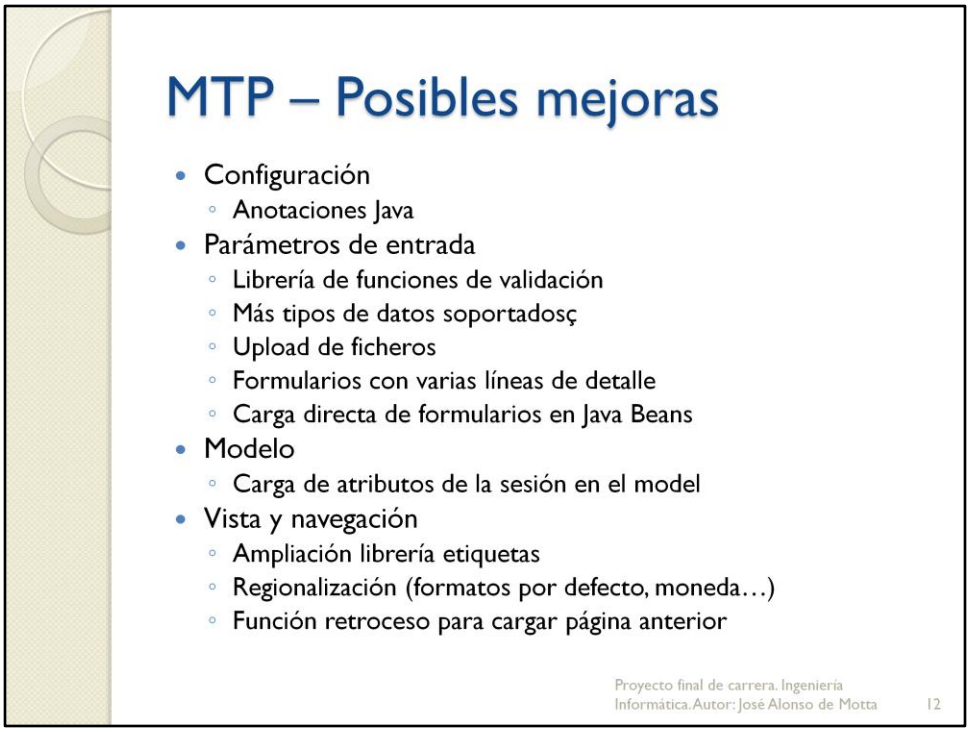
Las peticiones del usuario las recibe el Servlet controlador, que se encarga de encaminar el control de la petición en el siguiente orden:

- En primer lugar se **crea el objeto RequestData**, que contiene la información contextual de la petición. Este interfaz oculta a los demás componentes la implementación concreta del protocolo de comunicación con el cliente. Si no hay sesión de usuario creada, se hace en este paso.
- En segundo lugar se pasa el control al **ModelMapper**, que elige e instancia el modelo de aplicación que se ejecutará.
- Se verifica que la sesión tiene **permiso de ejecución** del modelo seleccionado, y se continúa el proceso o se redirige a la página de error de seguridad correspondiente.
- Después **se validan los parámetros** de entrada y se cargan a objetos Java con el tipo de datos adecuado. La verificación es de distintos tipos: conversión correcta al tipo de datos; campo obligatorio; función de verificación del parámetro; y función de verificación del conjunto de parámetros. Los parámetros se cargan en el modelo utilizando los métodos set. En caso de que la validación falle en algún punto, el framework envía el control a la vista de error de validación que concierna.
- Se procede a la **ejecución del modelo** (patrón Command), que invoca a los procesos de negocio correspondientes y carga los datos de salida para la siguiente vista.
- Se pasa el control al **ViewMapper** que, en función de la respuesta de la ejecución del modelo elige la siguiente vista a mostrar al usuario.
- Se **carga y ejecuta la vista** correspondiente, que será normalmente una página JSP, aunque también puede redirigir a otra petición del marco. Se devuelve el resultado al usuario.



El framework se divide en estos componentes:

- **Configurator**. Singleton que carga y contiene la configuración de la aplicación.
- **Controller**: Procesa las peticiones de los usuarios delegando las responsabilidades de las distintas fases del request en los helpers.
- **Helpers**. Su misión es apoyar al controller en diferentes aspectos. Contienen los algoritmos para gestionar parte del procesamiento de las peticiones.
    - **AccessControl**. Verifica que la sesión del request tiene permiso para ejecutar el modelo.
    - **ContextFactory**. Crea el objeto RequestData adecuado según el protocolo de comunicación del cliente. Se ha creado la implementación para el protocolo HTTP.
    - **ModelMapper**. Selecciona e instancia el modelo a cargar y ejecutar en función de la petición recibida.
    - **ModelRunner**. Ejecuta el modelo, encapsula las excepciones que pueda disparar y almacena la respuesta obtenida.
    - **Validator**. Verifica los parámetros de entrada, los transforma a los tipos de datos Java adecuados y carga los que sean necesarios en el modelo que actúa en esta fase como un Java Bean.
    - **ViewMapper**. Selecciona la vista a devolver al usuario en base a los resultados de las etapas de seguridad, validación y ejecuución del modelo.
- **RequestData**. Almacena la información de estado del request, la sesión y el modelo y actúa como interface entre la aplicación y el framework.
- **Taglib**. Librería de etiquetas que permite el acceso mediante tags desde las páginas JSP.



Configuración:

- Anotaciones Java

Parámetros:

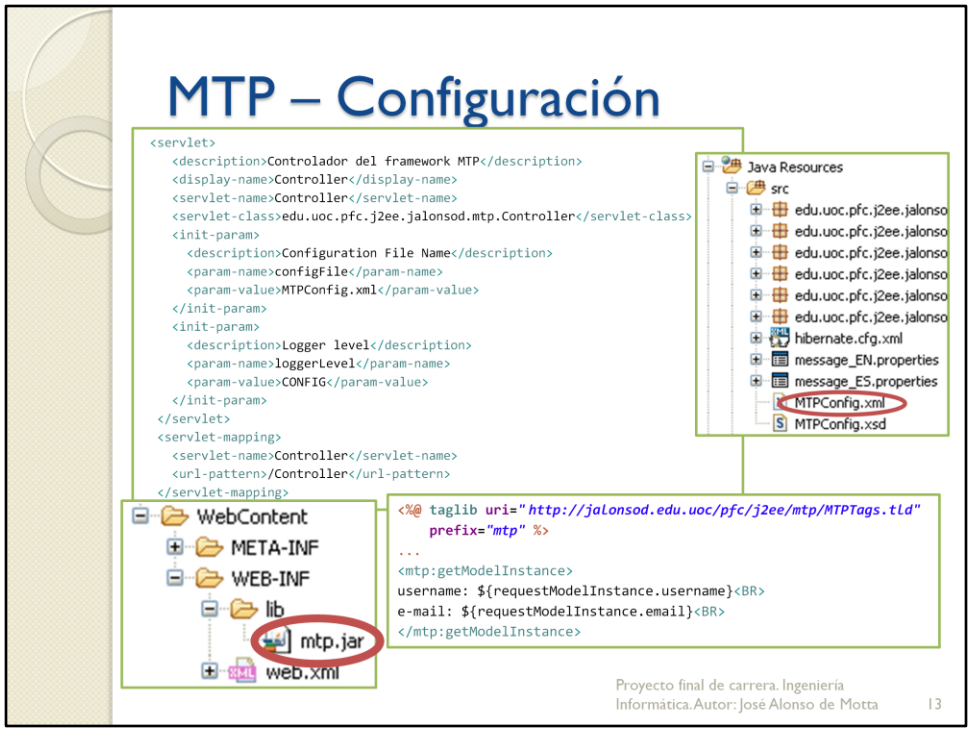
- Librería de funciones de validación
- Ampliación de los tipos de datos soportados
- Upload de ficheros.
- Inclusión de varias líneas de detalle en formularios (carga y validación de líneas de detalle en arrays de parámetros).
- Carga de formularios de entrada directamente en objetos tipo Java Beans.

Modelo:

- Carga de atributos de la sesión en propiedades del objeto Model.

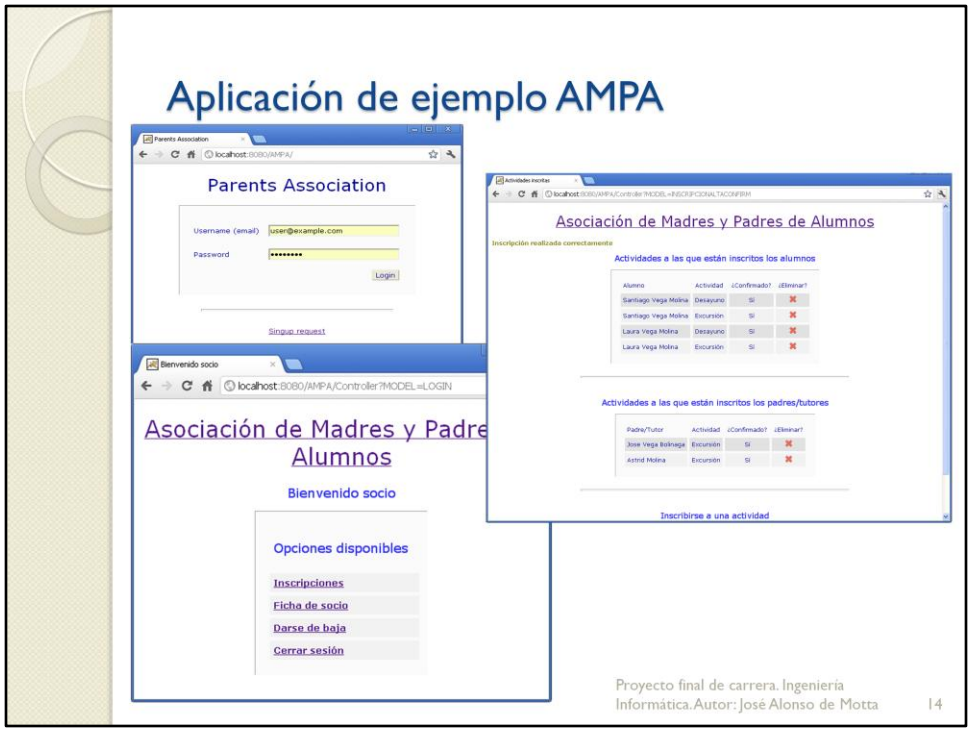
Vista:

- Integración con otras librerías de etiquetas o ampliación de la propia de MTP para incorporar componentes del interfaz de usuario, lo que permitiría realizar validaciones en cliente o incluir funcionalidad AJAX.
- Soporte de regionalización: diferentes representaciones de fecha/hora, moneda, separadores decimales y de millar…
- Incorporación de forma nativa de retroceso, para recargar la última página con los datos enviados y presentando los errores en caso de fallos de validación o autorización.



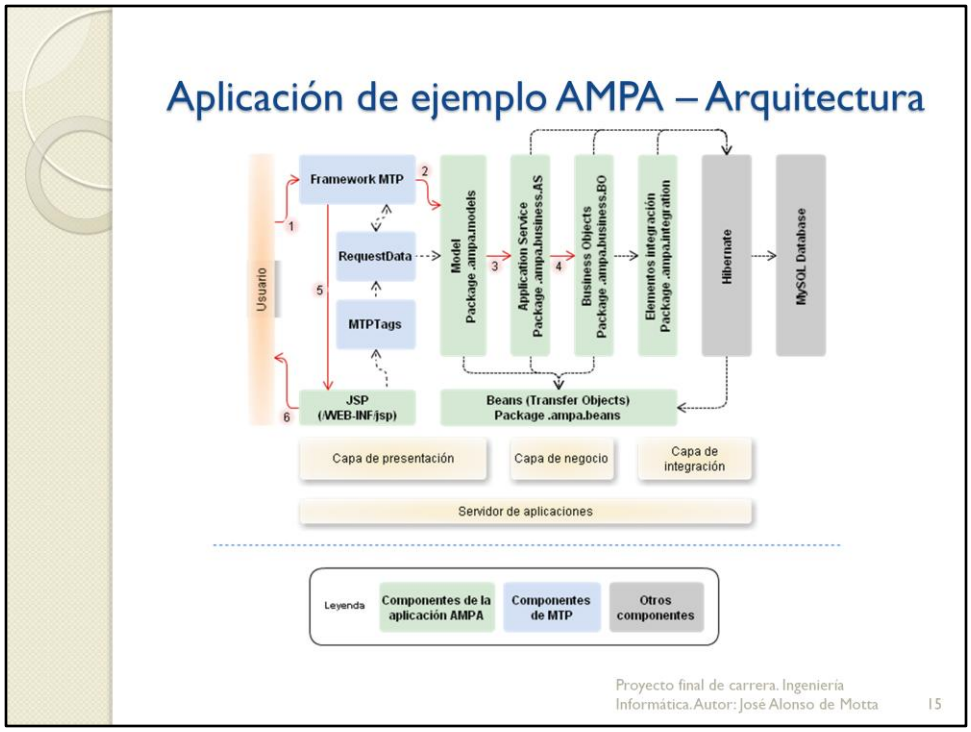
Para incorporar el framework a una aplicación, hay que seguir estos pasos:

- Incluir la **librería mtp.jar** en el classpath de la aplicación. Por ejemplo en el directorio WEB-INF/lib
- Definir el **Servlet** «Controller» de MTP en el **descriptor de despliegue** web.xml
- Construir el **fichero de configuración** (por defecto se llama MTPConfig.xml ubicado en la raíz de las clases) en base al schema MTPConfig.xsd incluido en la librería mtp.jar. Para recibir ayuda del IDE de desarrollo en la construcción del XML, puede ser necesario extraer el XSD de la librería y copiarlo en el proyecto.
- Si se desea soportar múltiples idiomas en la aplicación se pueden añadir **ficheros de propiedades** con las etiquetas del interfaz de usuario en distintos **idiomas**.
- Incluir en las páginas JSP que vayan a interactuar con el framework la directiva **taglib** para incluir la librería de etiquetas, e incorporar las etiquetas correspondientes en la página.



La aplicación de ejemplo pretende cubrir la gestión diaria de una **asociación de madres y padres de alumnos** (AMPA) con sus asociados, para que los socios puedan realizar algunas de las gestionar con la asociación a través de Internet.

En el proyecto se han implementado las funciones relativas a la ficha de socio e inscripción en actividades, con el objetivo de **demostrar y probar el funcionamiento del framework** así como de configurar una **arquitectura típica en tres capas** (presentación, negocio e integración) de una aplicación J2EE con interfaz web.



La imagen muestra, de izquierda a derecha, las distintas **capas de la aplicación**: presentación, negocio e integración.

Las **flechas rojas** indican los componentes por los que pasa el **control del request**.

Las **líneas de puntos negras** significan **invocaciones** a distintos elementos, pero que no suponen una entrega del control de la petición.

Los dos elementos que interactúan con el framework MTP, **el modelo y la vista** se sitúan en la capa de presentación.

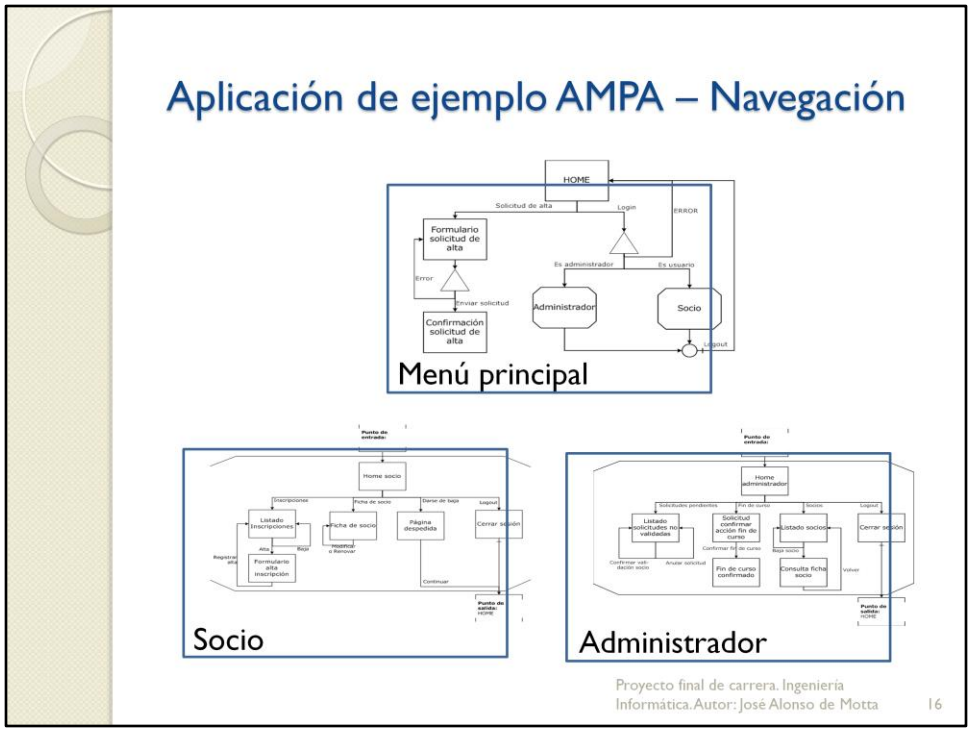
En la capa de negocio, la **lógica a nivel atómico** se lleva en los **Business Objects**, mientras que la **lógica a nivel de transacción** y la **fachada** de esta capa está en los **Application Service**.

Los **Beans** contienen los datos de negocio. Estos objetos tienen forma de Java Beans que actúan también como Transfer Objects y contienen anotaciones Java para indicarle al framework de persistencia Hibernate cómo acceder a la base de datos para utilizarlos.

La **capa de integración** está representada por el framework de persistencia Hibernate, para lo que se ha implementado también una clase en dicha capa. Como se ha dicho, los Beans también tienen anotaciones Java con el mapeo objeto-relacional, por lo que también pueden considerarse parte de la capa de integración.

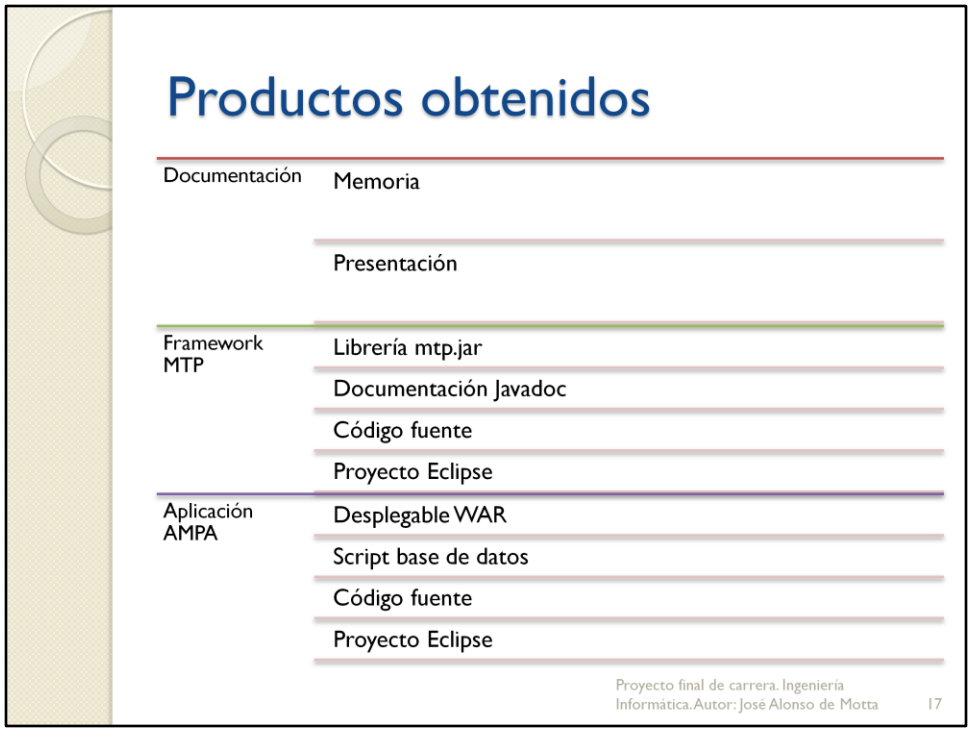
Todos estos elementos quedan empaquetados en una **aplicación web J2EE** (AMPA.war), y residen en un **servidor de aplicaciones J2EE**, en este caso Tomcat o JBoss.

La información de negocio se almacena en una **base de datos MySQL**, con la que dialoga la aplicación a través del framework de persistencia.



La aplicación desarrollada tiene tres **perfiles de usuario**:

- **No identificado**. El usuario que no se ha identificado únicamente tiene acceso a la página inicial (home) y a solicitar el alta como socio mediante el formulario correspondiente.
- **Socio**. Accede por la página home y se identifica mediante usuario y contraseña. Con ello accede al menú de opciones de socios. En la implementación se han desarrollado la modificación de la ficha de socio, la renovación de socio, la baja de socio y la inscricpión y baja en actividades organizadas por la asociación.
- **Administrador**. Accede por la misma página home que los demás, pero al identificarse con un usuario administrador, accede al menú de opciones de administradores. El administrador puede validar las solicitudes de socios pendientes, realizar la acción de fin de curso que invalida a todos los socios obligándoles a renovar, obtener un listado de socios, consultar su información y darlos de baja.

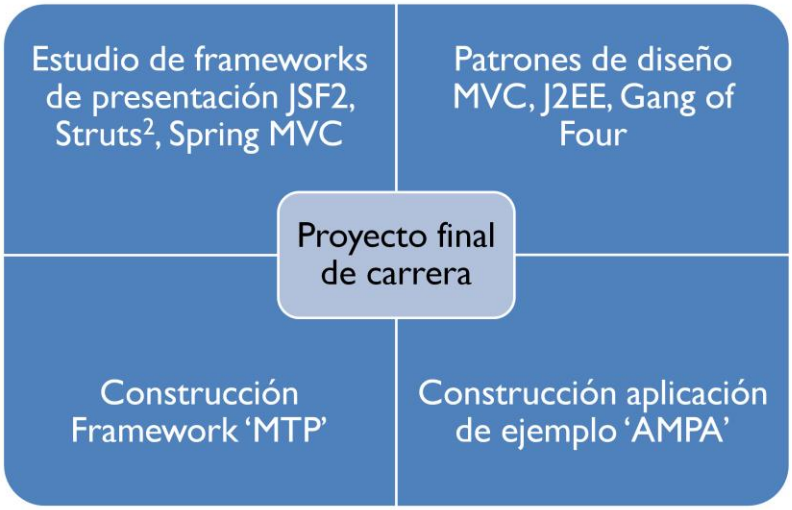


Como resultado del proyecto se han obtenido la memoria, esta presentación, el marco de trabajo y la aplicación de ejemplo.

El marco de trabajo se entrega para su distribución en una librería jar, la documentación del código en formato Javadoc, los programas fuente y el proyecto Eclipse (incluye las mismas fuentes) para su posterior modificación.

La aplicación de ejemplo se presenta en formato distribuible para su despliegue en un servidor de aplicaciones (ampa.war); el script para crear la base de datos y cargar los datos de ejemplo, el código fuente y el proyecto Eclipse (con las mismas fuentes).

Resumen


Estudio de frameworks de presentación JSF2, Struts2, Spring MVC
Patrones de diseño MVC, J2EE, Gang of Four
Proyecto final de carrera
Construcción Framework 'MTP'
Construcción aplicación de ejemplo 'AMPA'