
Servicio Andaluz de Salud
CONSEJERÍA DE SALUD

*Oficina de Calidad*
*Subdirección de Tecnologías de la Información*

# Catálogo de Normas de Desarrollo JEE

## Arquitectura de Referencia

Versión: v02r02
Fecha: 14/04/2011

# CONTROL DE CAMBIOS DEL DOCUMENTO

Registro de cambios

| Autor | Versión | Referencia de cambios | Fecha |
|---|---|---|---|
| Fco. Javier Delgado | v01r04 | Creación del documento | 10/11/2010 |
| Fco. Javier Delgado | v02r01 | Versionado. Sin cambios de contenido | 13/01/2011 |
| Fco. Javier Delgado | v02r02 | ✓ Ampliación del apartado 2.1.1 (Diseño por capas).<br>✓ Nuevo apartado 2.2.2.3.1 (Transaccionalidad)<br>✓ Actualización versiones tecnologías apartado 2.2.2.<br>✓ Renombrado del apartado 2.2.2.5 (Capa de Servicios e Integración).<br>✓ Actualización del apartado 2.2.2.6 (Capa de Servicios Transversales).<br>✓ Nuevos apartados 2.2.2.6.1 (Seguridad), 2.2.2.6.2 (Trazabilidad) y 2.2.2.6.3 (Instrumentación).<br>✓ Actualización apartado 3.2.<br>✓ Ampliación del apartado 4, 5 (Glosario, Referencias).<br>✓ Otros cambios menores. | 14/04/2011 |
| | | | |
| | | | |
| | | | |
| | | | |

Revisores

| Nombre | Versión Aprobada | Posición | Fecha |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |

Propiedades del documento

| Propiedad | Detalle |
|---|---|
| Título | Catálogo de Normas de Desarrollo JEE |
| Proyecto | Arquitectura de Referencia |
| Autor | Francisco Javier Delgado Leal |
| Nombre fichero | InfV1_JASAS_JEE_ArquitecturaReferencia_v02r02.doc |
| Fecha de creación | 08/11/2010 9:39 |
| Última actualización | 13/04/2011 11:07 |
| Plantilla base | |

# INDICE

# LISTA DE FIGURAS

# LISTA DE TABLAS

# 1 Introducción

La gran diversidad de tecnologías aplicadas hasta el momento para el desarrollo de las aplicaciones existentes en el marco de la Subdirección de Tecnologías de la Información (STI) del Servicio Andaluz de Salud (SAS) de la Junta de Andalucía (JA), hace necesaria la elaboración de unas normas específicas que posibiliten la homogeneización de los desarrollos y su adecuación al entorno de explotación específico definido.

## 1.1 Objetivo

Este documento **recoge la Arquitectura de Referencia (AR) a adoptar para los nuevos sistemas** de información JEE desarrollados por la STI.

Con la aplicación de estas normas y recomendaciones, la STI pretende:

- Homogeneizar la arquitectura de aplicaciones JEE.
- Documentar las distintas tecnologías que dan soporte a la arquitectura propuesta.
- Seleccionar un conjunto de tecnologías que conformen la AR.
- Normalizar otras tecnologías que, no siendo las propuestas, tienen interés por estar implantadas ya en la STI del SAS de la JA.

Con esto, el fin último de este documento es minimizar el impacto del crecimiento de la amplia diversidad de tecnologías empleadas en la actualidad en el desarrollo de sistemas de información, facilitando el crecimiento sostenible y reduciendo el impacto derivado del uso inapropiado de las tecnologías actuales.

## 1.2 Alcance

El presente documento va dirigido a los siguientes actores:

- Los Equipos de Desarrollo, que han de cumplir las normas y recomendaciones aquí presentadas.
- La Oficina de Calidad (OCA) y el equipo de Jefes de Proyectos del Departamento Desarrollo de la STI, que han de verificar su correcto cumplimiento.

## 1.3 Condiciones de Uso

Las normas y recomendaciones recogidas en este catálogo son de obligado cumplimiento. La STI podrá estudiar los casos excepcionales. Asimismo cualquier aspecto no recogido en este documento deberá regirse según las normas establecidas en el marco de desarrollo de la Junta de Andalucía (MADEJA), debiendo ser puesto de manifiesto ante la STI por el equipo de desarrollo.

La STI se reserva el derecho a la modificación de la norma sin previo aviso, tras lo cual, notificará del cambio a los actores implicados para su adopción inmediata.

En el caso de que algún actor considere conveniente y/o necesario el incumplimiento de alguna de las normas y/o recomendaciones, deberá aportar previamente la correspondiente justificación fehaciente documentada de la solución alternativa propuesta, así como toda aquella documentación que le sea requerida por la STI para proceder a su validación técnica.

# 2 Arquitectura de Sistemas de Información

En base a las necesidades de la STI, tomando como referencia los sistemas de información ya existentes tanto en el SAS como a nivel general en la JA, y teniendo en cuenta las normas actualmente existentes para las BBDD y los servidores de despliegue entre otros, se propone a continuación un modelo de arquitectura software a emplear para el desarrollo de nuevas aplicaciones JEE y la evolución, en su caso, de las ya existentes en proceso de desarrollo continuo.

## 2.1 Modelo Lógico

### 2.1.1 Diseño por capas

Para conseguir la independencia, robustez y escalabilidad de los sistemas, y en base a las pautas generales del modelado y diseño de los sistemas de información Web, **las aplicaciones desarrolladas por la STI deberán ser diseñadas según un modelo arquitectónico basado en capas lógicas**. Esta arquitectura deberá seguir, además, las pautas marcadas por el **Patrón MVC (Modelo-Vista-Controlador)**, empleando en cada una de las capas alguna de las soluciones tecnológicas expuestas en este documento.

De esta forma, como norma general, **los proyectos de la STI deberán modelarse, como mínimo, según una arquitectura tradicional basada en tres capas lógicas**:

- Capa de **Presentación y Control**.
- Capa de **Negocio**.
- Capa de **Persistencia**.

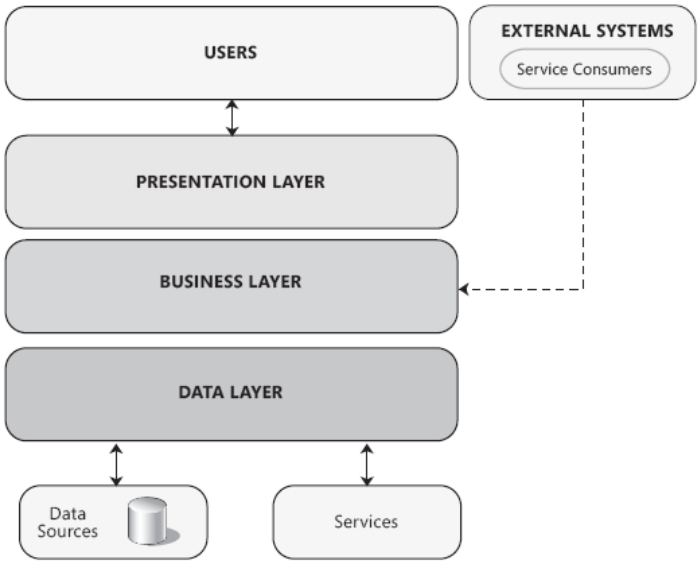


**Figura 1. Diseño lógico en 3 capas**

Este modelado en capas debe entenderse como básico y fundamental para el desarrollo de nuevas aplicaciones JEE, siempre en función de los requisitos específicos del sistema y de las necesidades del desarrollo a realizar.

De forma adicional, deben definirse otras capas necesarias:

- Capa de **Servicios e Integración.**
- Capa de **Servicios Transversales**, para englobar los servicios compartidos por todas las capas:
  - La **seguridad**

- La **trazabilidad**
- La **instrumentación**

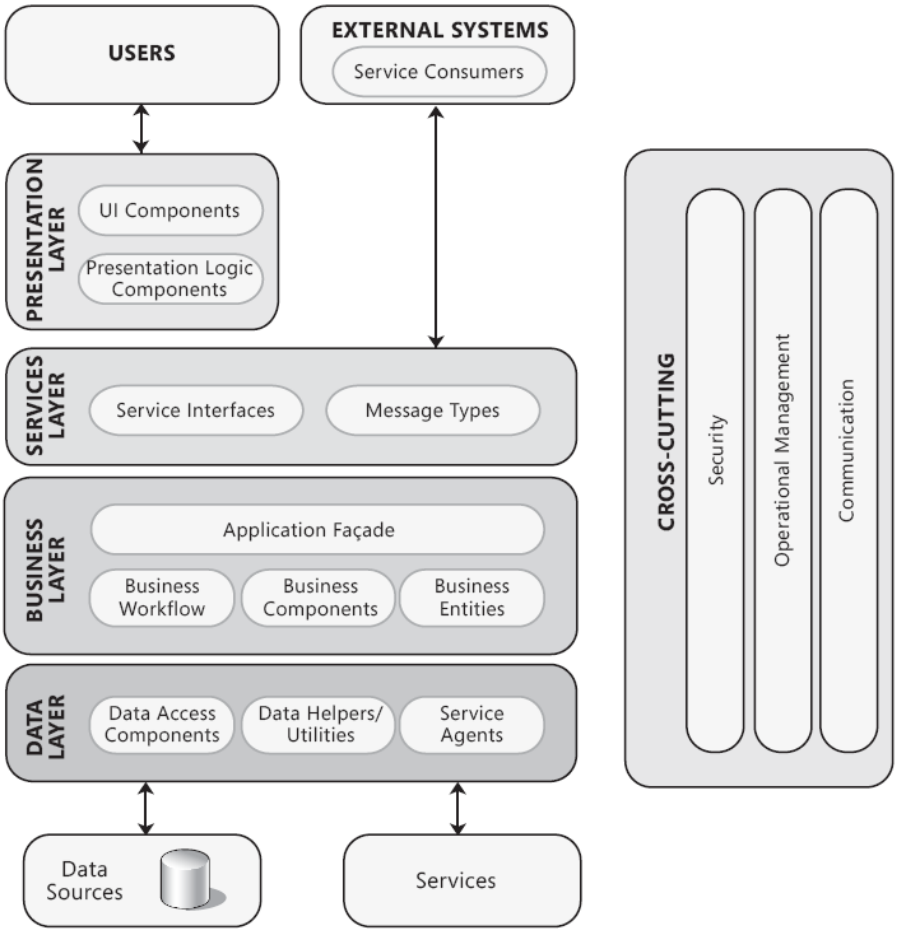


**Figura 2. Diseño lógico en 5 capas**

Si bien es posible la modificación de la arquitectura básica definida por las capas anteriores, **cualquier propuesta de cambio del modelo deberá ser presentada y justificada a la STI para su valoración y aceptación**.

La modularidad derivada de esta arquitectura en capas y que permite el desarrollo de las aplicaciones a varios niveles, posibilita a la aplicación de cambios evolutivos o correctivos de forma localizada sobre el nivel afectado. A este objetivo contribuye además el diseño de las capas en base a **Patrones de Diseño JEE** estándares junto con la aplicación de una serie de recomendaciones adoptadas como estándar de facto por la comunidad de desarrollo JEE.

### 2.1.2 Pautas de Diseño

Existen unas **pautas básicas** a considerar de forma general durante el diseño de las diferentes capas. Las principales se muestran a continuación.

#### 2.1.2.1 Capa de Presentación y Control

La Capa de Presentación muestra el sistema al usuario, presentándole la información y capturando aquella proporcionada por el usuario en un mínimo de proceso de filtrado. Esta capa se debe comunica únicamente con la capa de negocio.

Las pautas de diseño generales a adoptar en esta capa son:

- **Separación de la lógica de negocio y presentación**. La capa vista debe limitarse únicamente a la presentación de la información al usuario o a solicitársela, nunca a procesarla ni tomar decisiones sobre ella. Tales tareas serán competencia de las capas superiores.
- **Desacoplamiento entre capas.** La vista sólo debe contener lógica de visualización, nunca de tratamiento de la información, evitando de esta forma el acoplamiento entre capas.
- **Aspecto gráfico externo.** Todos los sistemas deben tener el mismo aspecto gráfico externo a nivel de proyecto, corporativo y/o de función de negocio. En general, el diseño de esta capa debe permitir la modificación del aspecto final (tanto estructura como estilo visual) sin necesidad de tener que modificar su código fuente asociado.
- **Modularidad.** Las vistas deben diseñarse de forma modular, evitando que cada módulo haga referencia a otros de la misma o de otras vistas. Es el caso, por ejemplo, de los módulos correspondientes a la cabecera, el cuerpo, el menú o pie de las vistas, que deberán integrarse dinámicamente en la misma página durante su ejecución. El grado de reusabilidad de las vistas así como el encapsulamiento de la funcionalidad en cada caso es muy elevado.
- **Internacionalización.** Las aplicaciones deben permitir su adaptación a diferentes idiomas y convenciones (formatos de fecha, moneda, etc.) sin necesidad de realizar cambios en su código.
- **Validaciones.** Las validaciones formales de los datos introducidos en las vistas, así como las correspondientes a campos obligatorios, deben ser realizados por la capa de presentación.
- **Conversiones de datos.** Las vistas recogen la información en forma de cadenas de caracteres, que deben ser convertidas a los objetos adecuados para su posterior tratamiento en capas superiores.
- **Evitar la lógica en la interfaz.** La vista se debe limitar a una secuencia de etiquetas que represente los controles gráficos. Cada uno de ellos tendrá una serie de atributos para configurar su aspecto (texto, color, estilo, tamaño, etc.) y su comportamiento (eventos, validaciones, conversores, etc.).
- **Elaborar un catálogo de controles.** Es recomendable elaborar un catálogo con la lista de controles oficiales en la que se especifique la función de cada control, los posibles validadores/conversores que se le puedan asociar, los eventos que pueda lanzar y los atributos que puedan cambiarse para manipular su aspecto externo. Este catálogo de controles facilitará la creación de las vistas de forma estándar e independiente de los componentes de la capa de negocio. De forma generalizada, no se recomienda la inclusión en este catálogo de controles desarrollados por terceros.
- **Gestión de los mensajes.** Los mensajes que las aplicaciones devuelvan deben proceder de los controles, del procesado de los datos de la vista (el resultado) o de la capa de negocio.

#### 2.1.2.2 Capa de Negocio

La capa de Negocio ocupa un lugar angular en la definición de una infraestructura de software base que permita el crecimiento y la extensibilidad de servicios para todas las aplicaciones existentes y futuras.

La definición de los límites de cada capa permite definir correctamente la colaboración que proveerá cada una de ellas. La ubicación de la capa negocios como capa intermedia provee de una infraestructura robusta y lista para la extensibilidad y el crecimiento como proveedora de servicios. Existen tres tipos de componentes de negocio fundamentales:

- Reglas de Negocio, que implementan la funcionalidad de negocio del sistema.
- Entidades de negocio, que representan las entidades del sistema.
- Workflow, para la implementación procesos de negocio en los cuales participan las entidades y la lógica de negocio.

Las principales pautas para el diseño de esta capa son:

- **Elementos constitutivos de la capa.** La capa de negocio engloba toda la lógica de negocio: reglas, workflow y operaciones no persistentes. Todas las operaciones persistentes deben ser delegadas a la capa de integración o persistencia.
- **Reglas de negocio.** La capa de negocio debe ser vista el punto de acceso único y centralizado al conjunto de servicios expuestos a la capa de presentación.
- **Validaciones de datos.** Los servicios de la capa de negocio deben realizar las comprobaciones necesarias sobre los datos proporcionados por la capa de presentación para garantizar su validación previa al inicio de los procesos de negocio requeridos.
- **Comunicación entre capas.** La definición de la estrategia de negocio, basada en la creación de las entidades del negocio o los objetos del modelo, se presenta como una fase fundamental en el desarrollo de esta capa. La comunicación entre capas se establece mediante estos objetos.
- **Transacciones.** El manejo de transacciones debe realizarse desde la capa de negocio, nunca desde la capa de persistencia, responsable de la provisión de los datos.
- **Tratamiento de excepciones.** Debe establecerse una gestión de excepciones según la cual las mismas se propaguen hacia la capa de presentación. Las excepciones generadas desde la capa de persistencia, y deben ser encapsuladas en esta capa para ser trasladadas de forma correcta a la capa superior.
- **Encapsular los servicios de negocio.** Los servicios de negocio son el "puente" entre el usuario y los servicios de datos. Responden ante peticiones del usuario (u otros servicios de negocios) para ejecutar una tarea de este tipo aplicando procedimientos formales y reglas de negocio a los datos relevantes. Cuando los datos necesarios residen en un servidor de bases de datos, garantizan los servicios de datos indispensables para cumplir con la tarea de negocios o aplicar su regla. Esto aísla al usuario de la interacción directa con la base de datos.
- **Control sobre el uso de servicios.** El control del acceso a los servicios de negocio debe hacerse en la capa de negocio, bien a nivel de servicio vertical (cada Fachada) o a nivel de método dentro de cada servicio.

#### 2.1.2.3 Capa de Persistencia

La capa de persistencia de la información es la parte más crítica de una aplicación software. En el caso en el que la aplicación haya sido diseñada en base a la orientación a objetos, la persistencia de los datos se consigue mediante la serialización de los objetos que los representan o mediante su almacenamiento en una base de datos, si bien actualmente las bases de datos más populares son relacionales.

El modelo de objetos difiere en muchos aspectos relacional. La interfaz que une ambos modelos se define en base a un marco de mapeo objeto-relacional. De esta forma, la capa de persistencia encapsula el comportamiento necesario para persistir objetos, entendiéndose como tal la realización de operaciones CRUD (Create, Read, Update, Delete) sobre el almacenamiento persistente o base de datos.

Al igual que para la capa de presentación y de negocio, para la capa de persistencia existen también unas **pautas básicas** a considerar de forma general durante su diseño. Las principales son:

- **Mapeo Objeto-Relacional (ORM).** La implementación de la capa de persistencia debe realizarse en base a un mapeo Objeto-Relacional, empleando un motor de persistencia que asegure el correcto funcionamiento de la capa.
- **Cada objeto debe ser creado en la base de datos para que sea persistente**. El patrón CRUD indica que cualquier objeto debe de ser creado como un elemento persistente en la base de datos. Es necesario asegurar que existen operaciones que permiten a la capa inferior leerlo, actualizarlo o simplemente borrarlo. Para ello, debe implementarse un DAO distinto por cada objeto de negocio en el sistema.

- **Manejo de la Cache.** El uso de cachés en el acceso a datos debe realizarse siempre teniendo en cuenta las características del sistema y las necesidades de sincronización. No hay que olvidar el coste de rendimiento derivado de su uso, por lo que el manejo de cachés deberá estar siempre justificado.
- **Permitir la concurrencia de usuarios.** La capa de persistencia debe permitir que múltiples usuarios trabajen en la misma base de datos protegiendo los datos de ser escritos equivocadamente, minimizando las restricciones en su capacidad concurrente para lectura y escritura.
- **Evitar las referencias circulares entre objetos.** En la medida de lo posible, se deben evitar las referencias circulares, facilitando la localización de los objetos, proporcionando un acceso más efectivo a la información.
- **Buen uso de la información oculta.** Existen numerosas columnas de tablas que no necesitan ser mapeadas a una propiedad del objeto, conteniendo información oculta para el modelo de objetos pero necesaria para el modelo relacional. En esta categoría entran los mecanismos de auditoría. Al leer el objeto, se lee esta información, que es ocultada al objeto pero mantenida por el framework de persistencia.
- **Actualización en cascada.** Debe utilizarse la posibilidad que ofrecen los frameworks para la actualización en cascada de objetos. Esta característica permite que las modificaciones hechas a un objeto se repliquen en los objetos relacionados, mejorándose su mantenimiento, la replicación eficiente de los cambios y la integridad de los datos.
- **Operaciones en masa.** Se deben de permitir las operaciones en masa sobre los datos almacenados, lo que mejora de forma sensible el rendimiento de las operaciones.

### 2.1.3 Patrones de diseño JEE

El diseño de las diferentes capas de una aplicación JEE debe estar basado, en la medida de lo posible y sin abusar de ello, en la aplicación de unos patrones de diseño específicos. Entendiendo como base imprescindible para un buen diseño el catálogo de patrones creacionales, estructurales y de comportamiento definidos en [GoF], la Figura 3 muestra la clasificación de patrones JEE según [JEE] catalogados por capas.

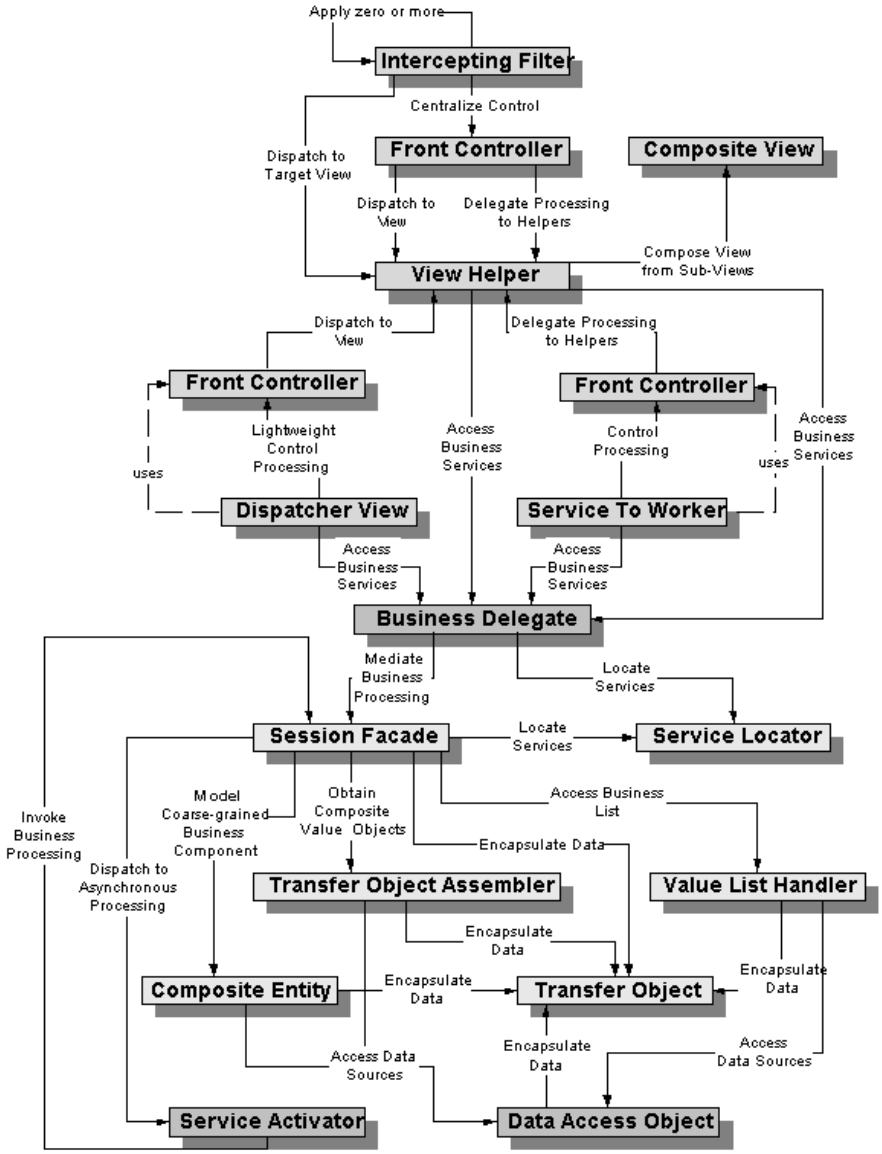


Figura 3. Patrones de Diseño JEE

2.1.3.1 Capa de Presentación y Control

Los patrones aplicables para el diseño de la capa de presentación son:

- **Intercepting Filter.** Un objeto ubicado entre el cliente y los componentes Web procesa las peticiones y las respuestas.
- **Front Controller.** Un objeto que acepta todos los requerimientos de un cliente y los direcciona a los manejadores apropiados.
- **View Helper.** Un objeto helper encapsula la lógica de acceso a datos en beneficio de los componentes de la presentación.
- **Composite View.** Un objeto vista compuesto de otros objetos vista.
- **Service to Worker.** Un objeto dispatcher que forma parte del Front Controller usa View Helpers a gran escala y ayuda en el manejo de la vista.
- **Dispatcher View.** Un dispatcher, que forma parte del Front Controller, no usa View Helpers y realiza muy poco trabajo en el manejo de la vista, siendo esta resposabilidad de los mismos componentes de la Vista.

#### 2.1.3.2 Capa de Negocio

Los patrones aplicables para el diseño de la capa de negocio son:

- **Business Delegate.** Un objeto que reside en la capa de presentación llama a métodos remotos en los objetos de la capa de negocios en beneficio del resto de componentes de la capa de presentación.
- **Transfer Object.** Un objeto serializable para la transferencia de datos sobre lar red.
- **Session Facade.** Un objeto de sesión actúa como fachada y maneja los objetos de negocio proporcionando un servicio de acceso uniforme a los clientes.
- **Composite Entity.** Un bean de entidad es construido o es agregado a otros beans de entidad.
- **Transfer Object Assembler.** Un objeto que reside en la capa de negocios crea Value Objets cuando es requerido.
- **Value List Handler.** Un objeto maneja la ejecución de consultas SQL, caché y procesamiento del resultado, generalmente implementado como bean de sesión.
- **Service Locator.** Un objeto abstrae todo el uso JNDI y oculta las complejidades de la creación del contexto inicial, de búsqueda de objetos home EJB y de recreación de objetos EJB.

#### 2.1.3.3 Capa de Persistencia

Los patrones aplicables para el diseño de la capa de persistencia son:

- **Data Access Object.** Un objeto abstrae y encapsula todos los accesos a la fuente de datos.
- **Service Activator.** Un objeto recibe peticiones y mensajes asíncronos de los clientes y localiza e invoca los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona.

## 2.2 Entorno Tecnológico

Estas recomendaciones también serán de aplicación para los desarrollos evolutivos y/o correctivos de sistemas ya desarrollados, si bien se debe tener en cuenta que para tecnologías anteriores (como PHP, Visual Basic, JSP, Servlets o JDBC) la inclusión o migración a las nuevas tecnologías no será siempre viable, pudiendo ser recomendable en muchos casos continuar con las tecnologías actuales del proyecto, como la elección de alguna tecnología diferente a las aquí descritas.

### 2.2.1 Tecnologías de Base y Requerimientos de Entorno

La arquitectura de referencia plasmada en este documento se basa en unas herramientas y tecnologías tomadas como base, y en las que se apoyarán muchas de las decisiones tecnológicas adoptadas. Será necesario por ello tener en cuenta las **Normas de Referencia** elaboradas por la STI para:

- El Servidor de Aplicaciones WebLogic.
- La Base de Datos Oracle.
- La Explotación de Datos.
- La Integración con el Bus ESB.

### 2.2.2 Tecnologías de Referencia

#### 2.2.2.1 Máquina Virtual Java

- Se debe utilizar la Máquina Virtual Java (JVM) de referencia para el Servidor de Aplicaciones WebLogic, según se especifique en el manual de normas correspondiente.
- En concreto, se debe utilizar la **versión 1.6** de la JVM.

#### 2.2.2.2 Capa de Presentación y Control

- Se debe emplear el estándar **JSF (JavaServer Faces)** para el diseño de la capa de presentación y control implementando el patrón MVC. En ambos casos, se debe usar la implementación de JSF de Oracle proporcionada por el propio contenedor Oracle WebLogic Server. En concreto, el desarrollo de la capa

de presentación debe realizarse con el estándar **JSF2.0 (JSR-314)**, haciendo uso de la implementación de Oracle **Mojarra 2.0.4.**

- De forma complementaria para simplificar las tareas de desarrollo, se recomienda el uso de las librerías de componentes **Richfaces** y **PrimeFaces**, en sus últimas versiones estables compatibles con las tecnologías empleadas: Richfaces **4.0.Final** y PrimeFaces **2.2.1**.
- Como librería adicional, en caso necesario, se recomienda el uso de la librería **jQuery,** en su última versión estable compatible (**1.5.2**) para eliminar, en caso necesario, el código JavaScript propio derivado de la creación de nuevos componentes con efectos visuales.

#### 2.2.2.3 Capa de Negocio

- La capa de negocio debe implementarse haciendo uso del framework **Spring**, en su versión **3.0.x** certificada para la versión actual de Oracle WebLogic Server. En concreto, la última versión estable del framework Spring certificada para la versión actual de Oracle WLS es la **3.0.5.RELEASE**.
- Para el mapeo de objetos se recomienda el uso de la librería **Dozer**, en su última versión estable (**5.3.2** o superior)**.**

##### 2.2.2.3.1 Transaccionalidad

- La transaccionalidad de las operaciones en la capa de negocio debe asegurarse mediante la implementación de **transacciones declarativas** haciendo uso de la orientación a aspectos proporcionada por **Spring AOP**.
- Debe utilizarse, en la medida de lo posible, un **enfoque basado en las anotaciones** proporcionadas por Spring usando, para ello, la anotación **@transactional**.
- Las anotación @transactional debe ser insertada antes de la definición de clases concretas o métodos, evitando en la medida de lo posible anotar con @transactional interfaces y definiciones de clases.

#### 2.2.2.4 Capa de Persistencia

- Se debe emplear el estándar **JPA2 (JSR-317)** para la implementación de la capa de Persistencia.
- Como implementación única del estándar debe utilizarse **EclipseLink**, en su versión **2.1.2**, proporcionada por el contenedor Oracle WebLogic Server**.**

#### 2.2.2.5 Capa de Servicios e Integración

- La integración de aplicaciones debe realizarse sobre el paradigma **SOA** (Arquitectura Orientada a Servicios).
- Como componente para la comunicación entre las diferentes aplicaciones deben utilizarse los **Servicios Web (WS)**.
- La implementación de estos Servicios Web debe realizarse siguiendo la especificación **JAX-WS,** en su versión **2.1.5,** provista por el contenedor de aplicaciones Oracle WebLogic. Se recomienda el uso del framework **Spring** para realizar la construcción de dichos servicios.
- Debe utilizarse el protocolo de mensajería **SOAP 1.2** para el formato de los mensajes, y el protocolo de transporte **HTTP/S**.
- Como norma general, se debe seguir la aproximación **Contract-First**, así como utilizar el mecanismo de codificación **document/literal** para la implementación de los mensajes SOAP, según el cuál el documento en sí constituye el cuerpo del mensaje.

#### 2.2.2.6 Capa de Servicios Transversales

##### 2.2.2.6.1 Seguridad

- El diseño de la seguridad a nivel de aplicación debe implementarse utilizando el framework **Spring Security**, en su versión **3.0.5.RELEASE** compatible.
- Por su parte, debe utilizarse **WS-Security 1.1** para dotar de seguridad a los Servicios Web desarrollados.

##### 2.2.2.6.2 Trazabilidad

- La trazabilidad de la aplicación debe implementarse haciendo uso del framework **Apache Log4J**, en su versión **1.2.15**, proporcionada por el contenedor.
- Para la declaración de los componentes de trazabilidad debe utilizarse el interfaz proporcionado por el framework **SLF4J (Simple Logging Facade for Java)**, en su última versión estable, en concreto la **1.6.1.**
- La implementación de la trazabilidad así como su configuración deben realizarse de forma tal que sea posible la modificación de los niveles de trazabilidad en caliente, sin necesidad de parar el sistema.
- Siempre y cuando sea posible, la trazabilidad de la aplicación debe configurarse en base a la **Programación Orientada a Aspectos (AOP)**, en tal caso, haciendo uso de los recursos proporcionados por **Spring AOP**.
- Los niveles de trazabilidad mínimos deben implementarse según la siguiente regla definida para los diferentes entornos existentes en la STI:
    - Entorno de **Desarrollo**: **ALL**.
    - Entorno de **Prueba**: **ALL**.
    - Entorno de **Preproducción**: **WARN**.
    - Entorno de **Producción**: **ERROR**.
- De forma adicional, y dependiendo de las necesidades y requerimientos de la funcionalidad de la aplicación, se recomienda la implementación de otros niveles de trazabilidad así como la creación de atributos específicos para ello, tal y como podría ser el caso de la trazabilidad de los casos de uso de la aplicación.
- No se permite, en ningún caso, insertar en los ficheros de log datos catalogados como de carácter confidencial según la LOPD.

##### 2.2.2.6.3 Instrumentación

- La instrumentación de las aplicaciones JEE debe implementarse con la tecnología **Java Management eXtensions (JMX)**, haciendo uso de la última versión estable proporcionada por el contenedor, en concreto, la **1.4.1**.
- Deben utilizarse las facilidades y el soporte proporcionado por el framework **Spring** para realizar la implementación de la instrumentación de forma **no intrusiva**.
- En cualquier caso, debe evaluarse el impacto del uso de esta tecnología en el rendimiento de la aplicación.

# 3 Resumen del Catálogo de Normas

## 3.1 Resumen de Pautas

| Pautas de Diseño | |
|---|---|
| **Norma** | **Carácter** |
| Diseño por Capas | Obligatorio |
| Patrones de Diseño JEE | Obligatorio |
| Patrón MVC | Obligatorio |
| **Capa de Presentación** | |
| Separación Lógica de negocio y Presentación | Obligatorio |
| Desacoplamiento entre Capas | Obligatorio |
| Aspecto gráfico externo | Obligatorio |
| Modularidad | Obligatorio |
| Internacionalización | Obligatorio |
| Validaciones | Obligatorio |
| Conversiones de datos | Obligatorio |
| Evitar la lógica en la interfaz | Obligatorio |
| Catálogo de controles | Recomendado |
| Gestión de mensajes | Obligatorio |
| **Capa de Negocio** | |
| Elementos constitutivos de la capa | Obligatorio |
| Reglas de negocio | Obligatorio |
| Validaciones de datos | Obligatorio |
| Comunicaciones entre capas | Obligatorio |
| Transacciones | Obligatorio |
| Tratamiento de excepciones | Obligatorio |
| Encapsulación de servicios de negocio | Obligatorio |
| Control sobre el uso de servicios | Obligatorio |
| **Capa de Persistencia** | |
| Mapeo Objeto-Relacional | Obligatorio |
| Persistencia de objetos | Obligatorio |
| Manejo de caché | Obligatorio |
| Concurrencia de usuarios | Obligatorio |
| Evitar referencias circulares entre objetos | Recomendado |
| Buen uso de la información oculta | Recomendado |
| Actualización en cascada | Obligatorio |
| Operaciones en masa | Recomendado |

**Tabla 1. Resumen de Pautas de Diseño de Aplicaciones JEE**

## 3.2 Resumen de Tecnologías

| Tecnologías por Capas | | |
|---|---|---|
| **Tecnología** | **Carácter** | **Versión** |
| JVM | Obligatorio | 1.6 |
| **Capa de Presentación** | | |
| Java Server Faces | Obligatorio | 2.0 |
| JSF2 | Obligatorio | 2.0 |
| Mojarra | Obligatorio | 2.0.4 |
| PrimeFaces | Recomendado | 2.2.1+ |
| Richfaces | Recomendado | 4.0.Final+ |
| jQuery | Recomendado | 1.5.2+ |
| **Capa de Negocio** | | |
| Spring | Obligatorio | 3.0.5.RELEASE |
| Dozer | Recomendado | 5.3.2+ |
| **Capa de Persistencia** | | |
| JPA2 | Obligatorio | 2.0 |
| EclipseLink | Obligatorio | 2.1.2 |
| **Capa de Servicios e Integración** | | |
| JAX-WS | Obligatorio | 2.1 |
| SOAP | Obligatorio | 1.2 |
| Spring | Recomendado | 3.0.5.RELEASE |
| **Capa de Servicios Transversales** | | |
| **Seguridad** | | |
| Spring Security | Obligatorio | 3.0.5.RELEASE |
| WS-Security | Obligatorio | 1.1 |
| **Trazabilidad** | | |
| Apache Log4J | Obligatorio | 1.2.15 |
| Simple Logging Facade for Java (SLF4J) | Obligatorio | 1.6.1+ |
| **Instrumentación** | | |
| Java Management eXtensions (JMX) | Obligatorio | 1.4.1 |
| Spring | Obligatorio | 3.0.5.RELEASE |

**Tabla 2. Resumen de Tecnologías por Capas**

# 4 Glosario

- **JA** – Junta de Andalucía
- **SAS** – Servicio Andaluz de Salud
- **STI** – Subdirección de Tecnologías de Información
- **OCA** – Oficina de Calidad
- **AR** – Arquitectura de Referencia
- **JEE** – Java Enterprise Edition
- **MADEJA** – Marco de Desarrollo de la Junta de Andalucía
- **Patrón de Diseño –** Esqueleto de la solución a un problema común en el desarrollo software que describe, a alto nivel de abstracción, una solución experta y extendida a un problema.
- **Patrón MVC (Modelo-Vista-Controlador) -** Patrón arquitectónico para el desarrollo de interfaces gráficas de usuario basado en la separación del modelo de la aplicación de su representación de cara al usuario y de la interacción de éste con la aplicación.
- **JSF2 (Java Server Faces 2) -** El estándar **JSF2** (JSR-314) evoluciona el estándar JSF incorporando mejoras sustanciales que simplifican el proceso de desarrollo. Así, los componentes propios junto con los de Facelets, ya incorporados al estándar JSF2, facilitan tanto la generación de nuevos componentes visuales por composición, así como su reutilización a modo de elementos estándar. JSF2 soporta directamente a tecnología AJAX (Asynchronous JavaScript and XML), reduciendo el número de peticiones completas realizadas al servidor, y mejorándose la usabilidad haciendo las páginas más dinámicas. Además, permite simplificar la complejidad derivada del uso de ficheros XML para la configuración tanto de la navegación existente en versiones anteriores gracias a la implementación de la navegación implícita y la configuración mediante anotaciones.
- **Mojarra –** Implementación de referencia de Oracle del estándar JSF.
- **PrimeFaces -** Librería de componentes para JSF que aporta, frente a los componentes estándar propios de JSF, una abstracción para el uso de la tecnología AJAX. No siendo PrimeFaces parte del estándar JEE, es la primera librería de componentes visuales que soporta de manera estable la versión 2 de JSF.
- **Richfaces -** Librería de componentes para JSF2. De forma similar a PrimeFaces, añade capacidad Ajax dentro de aplicaciones JSF existentes sin necesidad de recurrir al uso directo de JavaScript. Incluye ciclo de vida, validaciones, conversores y la gestión de recursos estáticos y dinámicos. Los componentes de Richfaces están construidos con soporte Ajax y un alto grado de personalización del "look-and-feel" que puede ser fácilmente incorporado dentro de las aplicaciones JSF.
- **jQuery –** Librería JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la tecnología AJAX a páginas Web, todo ello sin necesidad de generar código JavaScript propio. Primefaces se apoya en el uso extensivo de jQuery para la comunicación Ajax y el uso de componentes visuales, aunque también hace uso de la librería Javascript YUI de Yahoo.
- **Spring -** Framework de desarrollo para aplicaciones JEE que engloba un conjunto de funcionalidades muy importante para todas y cada una de las capas posibles del diseño de aplicaciones JEE. Su principal característica a destacar para la capa de negocio es su funcionamiento como contenedor de Inversión de Control (IoC - Inversion Of Control), aportando a los desarrollos la facilidad necesaria para integrar distintas piezas y servicios, minimizando su acoplamiento, y garantizando así un mejor mantenimiento y evolución de los diseños. Soporta además los estándares de inyección de

dependencias JSR-250 y JSR-330, y proporciona integración y facilidades de desarrollo con las tecnologías actuales más destacadas, como la gestión de seguridad, transaccionalidad, JSF, JPA, email, Servicios Web o EJBs, habiéndose convertido en un estándar de facto ampliamente extendido y aceptado en el mercado, y con una muy amplia comunidad de usuarios que lo soportan.

- **Dozer –** librería Java para el mapeo de Java Beans, copiando de manera recursiva los datos de un objeto a otro. Dado que, frecuentemente, estos Java Beans van a contener distintos tipos complejos. Dozer permite mapeos simples entre propiedades, mapeos complejos, mapeos bi-direccionales, mapeos implícitos y explícitos, y también mapeos recursivos, incluyendo éste el mapeo de los atributos de colecciones que también necesitan mapeos al nivel del elemento.
- **JPA2 (**Java Persistence API 2) - Estándar de Java (JSR-317) para el desarrollo con bases de datos relacionales. Facilita la abstracción de la fuente de datos, haciendo el código portable entre distintos sistemas gestores de bases de datos. Permite además centrar el desarrollo en la lógica de negocio, obviando los detalles de implementación la capa de acceso a datos, y posibilitando que los desarrollos sean más rápidos y más seguros.
- **EclipseLink -** Implementación del estándar JPA2.
- **SOA (Service Oriented Architecture) –** Paradigma de arquitectura de software que define la utilización de servicios para dar soporte a los requisitos del negocio. Permite la creación de sistemas altamente escalables que reflejan el negocio de la organización, y a su vez brinda una forma bien definida de exposición e invocación de servicios (comúnmente pero no exclusivamente servicios Web), lo cual facilita la interacción entre diferentes sistemas propios o de terceros. Con todo ello, SOA proporciona una metodología y un marco de trabajo para documentar las capacidades de negocio y puede dar soporte a las actividades de integración y consolidación.
- **JAX-WS (Java Api for XML Web Services**) - API Java para la creación de Web Services. JAX-WS forma parte del estándar Java EE. La implementación de referencia de JAX-WS es parte del proyecto GlassFish, y es de calidad productiva. En la plataforma Java EE 5 o superior, JAX-WS 2.0 reemplaza al API JAX-RPC. El cambio se basa en moverse hacia Web Services orientados a documentos (en vez de RPC).
- **SOAP (Simple Object Access Protocol) –** Protocolo estándar de mensajería para la implementación de la comunicación entre sistemas mediante Servicios Web.
- **Contract-First –** Aproximación de desarrollo de Servicios Web según la cual la generación del contrato WSDL es previa a la codificación del servicio.
- **WSDL (Web Services Description Language) –** Lenguaje de Descripción de Servicios Web.
- **Spring Security –** Marco para la creación de una capa de seguridad declarativa en aplicaciones basadas en Spring. Va más allá que la especificación JEE, proporcionando un marco más completo y totalmente independiente del entorno en el que se despliega la aplicación. Las condiciones de seguridad viajan en el artefacto a desplegar, lo que hace más fácil tanto el desarrollo como la migración o el mantenimiento del sistema, aportando más de independencia del entorno de despliegue del que se puede conseguir con JEE o JAAS. Spring Security aporta una solución completa para los dos principales requisitos de seguridad, la autentificación y la autorización, y lo hace tanto al nivel de peticiones Web, como a la hora de las llamadas a métodos. Para conseguir sus misiones Spring Security hace uso de las posibilidades de dependencia inyectada de Spring, las capacidades de AOP para la seguridad en la llamada a los métodos, y el uso de los filtros para la seguridad en las peticiones Web, usa lo mejor de Spring y lo mejor de JEE. De hecho, proporciona un puente entre ambos al permitir declarar los filtros y las dependencias de estos como componentes de Spring e inyectarlos en la aplicación como un componente más.

- **WS-Security (WebService Security) -** Conjunto de especificaciones destinadas a dotar a los servicios Web de los mecanismos de seguridad necesarios para su uso factible en ambientes en donde los criterios de autentificación de los usuarios e integridad de la información son características a las que es imposible renunciar. Sin WS-Security, la única manera de dotar de unos mínimos criterios de seguridad a los servicios Web, sobre todo en lo referente a la integridad de la información, es asegurando el canal de comunicación, normalmente con protocolo seguro SSH. Sin embargo, esto tiene un problema: lo que se protege es la comunicación punto a punto, pero si el mensaje va a viajar entre distintas máquinas, habría que volver a asegurar cada uno de los puntos, algo que no siempre es posible. De esta forma, la única manera de hacer que los servicios Web sean seguros es trasladar los artefactos de seguridad del canal al mensaje, definiendo los esquemas XML que dicen de una manera homogénea y unívoca qué partes de un mensaje SOAP están cifradas, cómo y con qué mecanismo, o qué credenciales se mandan para certificar al remitente. Esa es la función de WS-Security que, además, proporciona las utilidades para poder firmar y cifrar, en caso necesario, los documentos. Con todo esto, puede decirse que WS-Security no es una especificación, sino el paraguas común en el que se enmarcan cada una de las especificaciones que tratan temas de seguridad en WS, tales como WS-SecureConversation, WS-Federation, WS-Authorization, WS-Policy, WS-Trust, WS-Privacy, WS-Test, y a lo que hay que sumar otras como, XML Encryption, WS-I Basic Security Profile, SAML, XACML, X.509, etc.
- **SLF4J (Simple Logging Facade for Java)** – Framework Open Source que proporciona una pasarela a diferentes librerías de logging para encapsular el API de logging a utilizar proporcionando una única manera de implementar el registro de estados, testeo de usos y rastreo de errores sin tener en cuenta el API de log subyacente. De esta forma, se utiliza el API proporcionado por SLF4J en las declaraciones y el API del componente elegido en la ejecución.
- **Apache Log4J -** Framework Open Source desarrollado en Java por la Apache Software Foundation para permitir a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o "logs" (logging) en tiempo de ejecución y no en tiempo de compilación. La configuración de salida y granularidad de los mensajes es realizada en tiempo de ejecución mediante el uso de archivos de configuración externos.
- **JMX (Java Management eXtensions) –** Tecnología que proporciona una manera estándar y simple para administrar recursos tales como aplicaciones, dispositivos y servicios. Su especificación define una arquitectura dinámica, patrones de diseño, APIs y servicios que permiten la monitorización de los recursos mientras son creados, instalados e implementados. Para ello, provee a los desarrolladores Java de los medios para instrumentar el código, crear pequeños agentes Java, implementar administradores distribuidos e integrar fácilmente estas soluciones a los sistemas de administración y monitorización actuales.
- **Spring AOP (Aspect Oriented Programming) –** Framework utilizado para proporcionar servicios empresariales de manera declarative, especialmente como reemplazo para los servicios declarativos de EJB. El más importante de dichos servicios es en manejo declarativo de transacciones, que está construido sobre la abstracción de transacciones de Spring. Adicionalmente, AOP se usa en Spring para implementar aspectos personalizados, complementando así el uso de la Programación Orientada a Objetos.

# 5 Bibliografía y Referencias


- *ANEXO I -DGT- Requisitos básicos de entorno_1.005.pdf*
- Documentación *ATJ*
- Documentación de Proyectos
  - *SIGLO*
  - *RXXI*
  - *SIAGC*
  - *DAE*
- [OoO] ***Wrox Press Expert One-On-One j2Ee Development Without EJB***. Rod Johnson and Juergen Hoeller. Wiley Publishing, Inc. 2004
- [GoF] ***Design Patterns: Elements of Reusable Object Oriented Software***. Gamma E., Helm, R., Johnson, R., Vlissides J. Addison Wesley, 1995.
- [JEE] ***Core J2EE Patterns: Best Practices and Design Strategies***. Deepak Alur, John Crupi and Dan Malks. Prentice Hall / Sun Microsystems Press. 2nd Edition, June, 2003
- http://madeja.i-administracion.junta-andalucia.es/madeja/
- http://www.oracle.com/
- http://www.springsource.org/
- https://javaserverfaces.dev.java.net/
- http://jquery.com/
- http://www.primefaces.org/
- http://www.jboss.org/richfaces
- http://jcp.org/en/jsr/
- http://www.eclipse.org/eclipselink/
- http://www.slf4j.org/
- http://logging.apache.org/log4j/