

DDT Cobro Modelo N28

Documento de Diseño Técnico

**Archivo: ERU Cobro Modelo N28.docx**

**Versión: 1**

**Equipo: Pasarelas de Pago**

**Fecha: 13/08/18**

# Índice

[Índice 2](#_Toc522271304)

[1. Descripción del documento 3](#_Toc522271305)

[1.1. **Referencias** 3](#_Toc522271306)

[1.2. **Definiciones y acrónimos** 3](#_Toc522271307)

[1.3. **Control de modificaciones sobre el documento** 3](#_Toc522271308)

[2. Introducción 4](#_Toc522271309)

[2.1. Objetivo 4](#_Toc522271310)

[3. Arquitectura de la Aplicación 5](#_Toc522271311)

[3.1. Arquitectura general 5](#_Toc522271312)

[3.2. Contexto Tecnológico 5](#_Toc522271313)

[3.3. Arquitectura Interna 6](#_Toc522271314)

[3.4. Versiones de las Librerías 9](#_Toc522271315)

[4. Organización Interna de los Módulos 11](#_Toc522271316)

[5. Módulo de Cobro N28 13](#_Toc522271317)

[5.1. Funcionalidad 13](#_Toc522271318)

[5.2. Diseño 13](#_Toc522271319)

[5.3. Transiciones entre Estados 14](#_Toc522271320)

[6. Módulo de Cifrado de Datos 15](#_Toc522271321)

[6.1. Funcionalidad 15](#_Toc522271322)

[6.2. Diseño 15](#_Toc522271323)

[6.3. Transiciones entre Estados 19](#_Toc522271324)

[6.4. Importación y Exportación de Operaciones 19](#_Toc522271325)

[7. Integración con Sistemas Externos 22](#_Toc522271326)

[7.1. Integración con Documentum 22](#_Toc522271327)

[7.2. Integración con @Firma 23](#_Toc522271328)

[8. Ficheros de Configuración de la Aplicación 24](#_Toc522271329)

[9. Convenciones del Desarrollo 26](#_Toc522271330)

[9.1. Estructura del Proyecto 26](#_Toc522271331)

[9.2. Definición de los Paquetes de la Aplicación 26](#_Toc522271332)

[9.3. Nombres de Clases, Métodos y Variables 26](#_Toc522271333)

[9.4. Gestión de Excepciones 27](#_Toc522271334)

[9.5. Plantillas y Ensamblado de las páginas 27](#_Toc522271335)

[9.6. Convenciones para los Objetos de la Base de Datos 28](#_Toc522271336)

[10. Arquitectura Física y Despliegue 30](#_Toc522271337)

[10.1. Topología de la Plataforma 30](#_Toc522271338)

[10.2. Despliegue en los entornos de Pre-Producción y Producción 30](#_Toc522271339)

[11. Entorno de Desarrollo 32](#_Toc522271340)

[11.1. Herramientas 32](#_Toc522271341)

[11.2. Versiones de las Herramientas 34](#_Toc522271342)

[11.3. Ciclo de Desarrollo con SVN y Maven 34](#_Toc522271343)

# Descripción del documento

Este documento refleja las necesidades de negocio relativas a un sistema de información (en adelante, sistema).

El documento trata de describir de forma rigurosa las necesidades del negocio y para ello trata evitar el manejo de conceptos ajenos a su ámbito en la definición de estas necesidades (requisitos de usuario).

El objetivo del documento consiste en formalizar los requisitos funcionales y de aquellas capacidades adicionales (características no funcionales) que ha de suministrar el sistema.

Este documento será utilizado por los Clientes de la Unidad de Organización y Sistemas para validar si el contenido del mismo coincide con sus necesidades manifestadas. También será utilizado por las personas de la Gerencia de Sistemas responsables de efectuar el desarrollo correspondiente.

* 1. **Referencias**

| **Título** | **Autor** | **Fecha** | **Editor** | **Fuentes** |
| --- | --- | --- | --- | --- |
| ERU Cobro Modelo N28.docx | Esteban Martín-Tembleque Poves | Junio 2018 |  |  |

Lista completa de los documentos referenciados en el ERU (Especificación de Requisitos de Usuario)

Identificación de cada documento por medio de un título, un número de informe, fecha y el organismo que lo ha publicado

Fuentes para conseguir los documentos referenciados

* 1. **Definiciones y acrónimos**

Definiciones de todos los términos, abreviaturas y siglas precisas para interpretar correctamente el documento.

Esta información puede proporcionarse haciendo referencia a uno o más apéndices o a otros documentos

| **Término** | **Definición** |
| --- | --- |
| CARM | Comunidad Autónoma de la Región de Murcia |
| CCT | Código de Control Tributario |
| MAC | Código de Control de verificación de campos |

* 1. **Control de modificaciones sobre el documento**

Identificación de las modificaciones hechas sobre el documento, indicando la fecha, la versión sobre la que se realizó el cambio, los capítulos afectados, una breve descripción del cambio y el autor o responsable de la modificación.

| **Fecha** | **Versión** | **Capítulo Afectado** | **Observaciones** | **Autor Modificación** |
| --- | --- | --- | --- | --- |
| 16/08/2018 | 1.0 | Todos | Versión inicial | Esteban Martín-Tembleque Poves |

# Introducción

## Objetivo

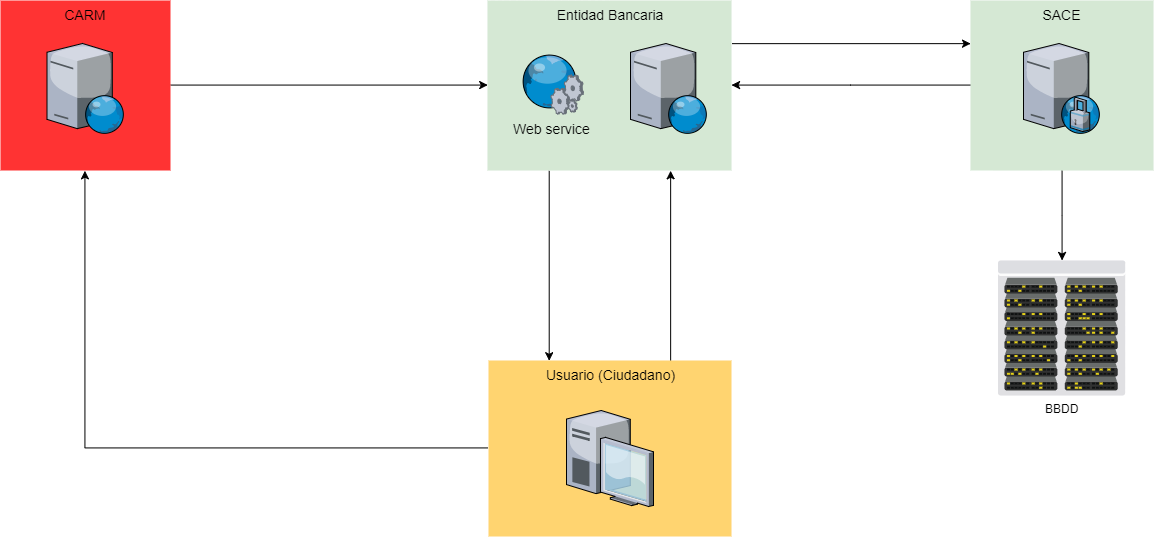
El objetivo fundamental del documento que se presenta a continuación es la realización de un diseño técnico pormenorizado de los procesos necesarios para la implementación de la aplicación del Cobro del Modelo N28 para la CARM (Comunidad Autónoma de la Región de Murcia).

Así, este documento tiene por objeto la recopilación del diseño técnico de la aplicación para guiar al equipo de desarrollo durante la fase de implementación.

# Arquitectura de la Aplicación

## Arquitectura general

El siguiente diagrama muestra la arquitectura general del sistema. De un lado están los usuarios que se conectan a la página de tributos de la CARM y desde allí son redirigidos a la página de introducción de datos de la pasarela de pago de la entidad bancaria. La Entidad Bancaria se comunica con el SACE para la gestión del pago y su inclusión en la BBDD. La interacción entre los actores y módulos de CARM y los módulos de la entidad bancaria se realizará a través de una aplicación web e invocación de servicios de negocio.



Los siguientes apartados abordan la arquitectura interna de los módulos de la entidad bancaria (aplicaciones y servicios) y su implementación.

## Contexto Tecnológico

En el diseño de las aplicaciones distribuidas, se ha convertido en un principio ampliamente aceptado la división de cada aplicación en componentes que ofrezcan los servicios de presentación, lógica de negocio e integración. Los componentes que realizan los mismos tipos de funciones se pueden agrupar en capas que están organizadas en una jerarquía o pila. Así, los componentes que están en una misma capa interactúan entre ellos directamente y con los componentes de las capas inferiores—pero nunca invocan a los componentes de las capas superiores.

Es importante tener en cuenta que las capas son simplemente agrupaciones lógicas de los componentes de software que conforman la aplicación o servicio. Las capas ayudan a diferenciar entre los tipos de tareas que realizan los componentes y facilitan la comunicación del diseño y la reutilización en la solución.



La siguiente tabla resume las características de cada una de las capas. Los apartados que vienen a continuación las describen con más detalle.

|  |  |  |
| --- | --- | --- |
| Capa | Descripción | Componentes |
| **Presentación** | Implementa las páginas de la interface gráfica de los usuarios. Es la cara visible de la aplicación. | Páginas JSP, Actions (Struts), plantillas HTML, CSS |
| **Negocio** | Implementa la lógica del funcionamiento de la aplicación, gestión de las transacciones, conversiones de tipos de datos. Es el corazón de la aplicación. | Objetos de negocio y servicios implementados con clases Java y que se integran con Spring |
| **Integración** | Gestiona el acceso a todos los sistemas externos: SACE | SNG de acceso a componentes de negoco Bankia |

###### Capa de Presentación

En el caso de las aplicaciones, la capa de presentación se encarga de implementar la interacción con los usuarios finales. Se implementa con Struts, páginas JSP, código HTML y Javascript. Aquí también se instalan los componentes de @Firma: la integración del MiniApplet de @Firma (cliente ligero para la firma digital en navegadores Web), permite ahorrar tiempo y esfuerzo en otros mecanismos de firma digital que supongan un aumento en la carga y costes de desarrollo y pruebas.



Componentes y librerías utilizados para implementar la capa de presentación

El siguiente diagrama de clases muestra una parte de la jerarquía de actions (Struts) de la capa de presentación de la aplicación web. Nótese que la superclase AplicacionAction tiene métodos que simplifican la gestión de la sesión y del usuario. Estos métodos están disponibles en todas las subclases y se utilizan para gestionar la seguridad, el rol actual del usuario, su idioma actual, etc.



Los *actions* utilizan clases auxiliares para mostrar y editar la información de los formularios de las páginas JSP. El siguiente diagrama muestra algunas de estas clases:



###### Capa de Negocio

La capa de negocio está compuesta objetos de negocio (beans de Java) y un conjunto de servicios. Los objetos de negocio son las entidades identificadas en la aplicación, como por ejemplo: las operaciones, los indicadores, los beneficiarios. Los servicios se encargan de gestionar las transacciones y coordinan el flujo de las operaciones que se inician con las acciones de los usuarios desde la capa de presentación.

Esta capa delega los accesos a la base de datos en la capa de persistencia y las invocaciones de servicios remotos a la capa de integración. Finalmente, aquí también se genera los reportes (con POI y JasperReports).



Componentes y librerías utilizados para implementar la capa de servicios

###### Capa de Integración

Esta capa gestiona las invocaciones a los servicios remotos (Documentum, @Firma). Los servicios se invocan mediante stubs generados con Apache CFX.



Componentes y librerías utilizados para implementar la capa de integración

El gestor documental Documentum proporciona una API de Java que permitiría realizar tareas específicas desde el propio aplicativo. Documentum puede emplearse para la gestión de los documentos que deben aportarse en los módulos de Justificación Financiera, Verificación y Auditoría.

Otro componente externo importante es el servidor de correo que se utiliza para el envío de todas las notificaciones a los usuarios (activación de cuentas, envío de mensajes y alertas de operaciones, etc.). Este servidor de correo está proporcionado por el MECD (host = servidor.mecd.es, puerto = 25). Mediante la integración de envío de correos de Spring se accede a este servidor para el envío de estas notificaciones.

Finalmente, es importante señalar que la integración con el Registro de Centros de Formación del MECD se realiza a través de la capa de persistencia porque la consulta del Registro se hace con una vista creada en la base de datos: MVW\_CENTROS\_WEB\_V1.

## Versiones de las Librerías

La siguiente tabla resume las versiones de las principales librerías utilizadas en la implementación de los módulos.

Nótese que la tabla no incluye todas las dependencias requeridas ni las librerías auxiliares (Apache Commons, Spring Security, etc.).

|  |  |
| --- | --- |
| Librería | Versión |
| CFX | 2.7 |
| Hibernate | 4.1 |
| Spring | 3.1 |
| Struts | 2.3 |
| Log4J | 1.2 |
| POI (generar y procesar Excel) | 3.11 |
| JasperReports (reportes en pdf) | 5.6.1 |
| Driver Oracle | 6.x (compatible con Oracle 11g) |

# Organización Interna de los Módulos

En el apartado anterior se presentaba la siguiente jerarquía de capas “lógicas”: Capa de Presentación, Capa de Negocio y Capa de Persistencia:



Las capas implementan diferentes aspectos de los módulos funcionales de la aplicación y por lo tanto pueden verse que son transversales a éstos. El siguiente diagrama muestra algunos módulos de la aplicación. Nótese que todos los módulos tienen requerimientos de implementación en las tres capas de la arquitectura:



Las capas se integran entre sí mediante el mecanismo de inyección de dependencias de Spring (anotación @autowired).

El siguiente diagrama muestra la cadena de delegación de responsabilidades desde un componente de la capa de presentación (OperacionesFPBAction) en un servicio de la capa de negocio (OperacionesService) y éste a su vez delega en un repositorio de la capa de persistencia (OperacionesRepository):



El siguiente diagrama de secuencia muestran el flujo completo de invocaciones desde la capa de presentación (páginas HTML) hasta la capa de persistencia:



Los siguientes apartados describen los detalles de la implementación de este flujo en cada una de las capas

# Módulo de Cobro N28

## Funcionalidad

El módulo de usuarios implementa toda la gestión de los usuarios de la aplicación:

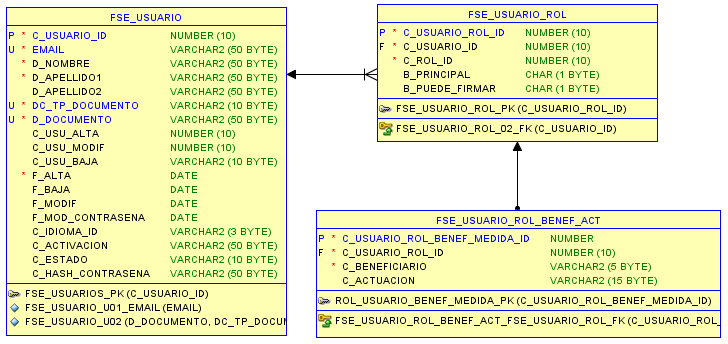
* Alta de nuevos usuarios en la aplicación
* Modificación de los datos de un usuario
* Baja de un usuario existente. Nótese que todas las bajas de usuarios son “lógicas”—los usuarios no son borrados físicamente de las tablas
* Mecanismo de activación de los usuarios que utilizan contraseña para acceder—envío de correos
* Recuperación de contraseñas

## Diseño

El siguiente diagrama de clases muestra las entidades de la capa de negocio que participan en este módulo:



El siguiente esquema muestra las tablas utilizadas por el módulo de gestión de usuarios:



## Transiciones entre Estados

Durante su ciclo de vida, los usuarios van cambiando de estado. Dentro de la aplicación, los estados se representan con un tipo enumerativo: **EstadoUsuario**. El siguiente diagrama resume los principales estados de los usuarios:



# Módulo de Cifrado de Datos

## Funcionalidad

El módulo de cifrado implementa todas las operaciones de cifrado/descifrado de datos en la aplicación:

* Alta, baja y modificación de operaciones FPB
* Alta, baja y modificación de operaciones ESO
* Alta, baja y modificación de operaciones Acredita
* Búsqueda de operaciones utilizando filtros
* Exportación de operaciones a Excel
* Importación de operaciones desde Excel
* Presentación, validación y devolución de operaciones

## Diseño

El siguiente diagrama de clases muestra las entidades de la capa de negocio que participan en este módulo:

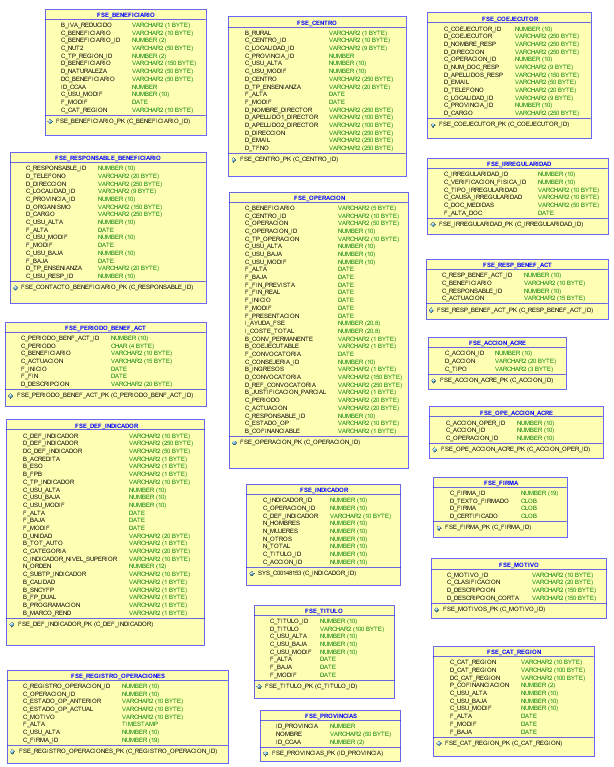


Las clases que aparecen en el centro y están pintadas de color más oscuro implementan la funcionalidad de las operaciones. La superclase de la jerarquía, **Operacion**, es abstracta e implementa el mecanismo genérico para la gestión de los indicadores colaborando con las clases **DefinicionIndicador**, **Indicador**, **TipoIndicador**). Las subclases se basan en este mecanismo y lo especializan para implementar la gestión de los indicadores para cada tipo específico de operación (FPB, ESO, Acredita, etc.). Así, por ejemplo, las operaciones de FPB tienen indicadores a nivel de los títulos y las operaciones de ESO solo tienen indicadores a nivel de centro de formación.

En modelo de clases también incluye tipos enumerativos que representan conceptos como los tipos de operaciones, las actuaciones, objetivos, prioridades y ejes:



El siguiente esquema muestra las tablas utilizadas por el módulo de operaciones:



## Transiciones entre Estados

Durante su ciclo de vida, las operaciones van cambiando de estado. Dentro de la aplicación, los estados se representan con un tipo enumerativo: **EstadoOperacion**.

En general, estos cambios son iniciados por el usuario cuando utiliza la aplicación e invoca opciones especiales de menú—solo es accesibles con roles específicos de los usuarios. Algunas transiciones provocan el envío de notificaciones de correo a los actores involucrados en la operación (responsable, etc.).

También hay transiciones automáticas que se producen con el paso del tiempo y que son iniciadas por procesos internos de la aplicación. El siguiente diagrama resume los principales estados de las operaciones:



## Importación y Exportación de Operaciones

El módulo de operaciones también incorpora funcionalidad para importar y exportar operaciones. Los siguientes diagramas de secuencia muestran el flujo completo de invocaciones para la exportación de operaciones de tipo ESO y FPB respectivamente:





Los siguientes diagramas de secuencia muestran el flujo completo de invocaciones para la validación e importación de operaciones de tipo FPB. El flujo para las operaciones de tipo ESO es similar—solo varían las plantillas de datos.



# Integración con Sistemas Externos

## Integración con Documentum

La aplicación permite a los usuarios adjuntar documentación en varios de los módulos, como por ejemplo adjuntar las actas de las verificaciones in-situ. Por otro lado, la aplicación también genera documentos que son descargados por los usuarios, como ocurre con las solicitudes de reembolso en el módulo de justificaciones y gastos.

En ambos casos, toda la documentación recibida y producida por la aplicación es almacenada en un gestor documental: Documentum.

Cada documento almacenado en Documentum tiene las siguientes propiedades:

* Un ID para identificarlo unívocamente. Este ID permite a la aplicación descargar el documento del gestor documental y mostrarlo al usuario
* Un contenido binario (secuencia de bytes)
* Varios metadatos asociados que se representan con tipos documentales y otros atributos (fecha de creación, etc.)

La aplicación interactúa con Documentum a través de un servicio web desplegado en la intranet del MECD. El siguiente diagrama muestra las principales clases que participan en la integración:



## Integración con @Firma

La aplicación también se integra con @Firma para ofrecer la siguiente funcionalidad:

* Validar los certificados de los usuarios que se identifican con un certificado digital. Este paso requiere que la aplicación se conecte a un servicio web desplegado en la intranet del MECD.
* Realizar la firma electrónica de la documentación presentada por los usuarios. Para esto se utiliza el mecanismo de firma implementado por el MiniApplet de @Firma.

El MiniApplet de @Firma es una herramienta de firma electrónica que funciona en forma de Applet de Java integrado en una página Web mediante JavaScript. Este applet no almacena ningún tipo de información personal del usuario, ni hace uso de cookies ni ningún otro mecanismo para la gestión de datos de sesión. El MiniApplet sí almacena el log de su última ejecución a efectos de ofrecer soporte al usuario si se encontrase

algún error durante la ejecución del Applet.

La integración del MiniApplet se ha realizado según las indicaciones del correspondiente manual: “Manual del integrador del MiniApplet v1.3 del Cliente @Firma” disponible en:

<http://administracionelectronica.gob.es/ctt/afirma/>

El siguiente diagrama muestra las principales clases de la aplicación que participan en la integración con @Firma:



# Ficheros de Configuración de la Aplicación

La siguiente tabla muestra todos los ficheros de configuración de la aplicación. Todos ellos van empaquetados en el fichero de despliegue (.war).

|  |  |
| --- | --- |
| Nombre del fichero | Descripción |
| **datasource.xml** | Contiene los parámetros de la conexión a la base de datos. Existen 4 versiones de este fichero, una para entorno de despliegue. Al momento de generar el **war**, Maven inserta la versión que corresponda al entorno de despliegue seleccionado. Ver más detalles en: **notas.html** en el proyecto Eclipse. |
| **gesfse.properties** | Contiene varios los parámetros de configuración utilizados internamente por la aplicación y que no son específicos de ninguna librería. Ver el listado de parámetros en el siguiente apartado. |
| **persistencia.xml** | Declara parámetros que se utilizan para configurar la capa de persistencia y la integración de Spring con Hibernate |
| **presentacion.xml** | Declara parámetros que se utilizan para configurar la capa de presentación: actions de Struts, rutas de seguridad, integración con Spring |
| **servicios.xml** | Declara parámetros que se utilizan para configurar la capa de negocio, la integración con Spring, CXF y gestión declarativa de las transacciones |
| **webservices.xml** | Declara los servicios web consumidos por la aplicación (@Firma, etc.) |
| **\*.vm** | Son las plantillas utilizadas por Velocity para crear el contenido HTML de los correos de activación de usuarios y recuperación de las contraseñas. |
| **messages\_\*.properties** | Son las plantillas con la traducción de los mensajes y etiquetas mostrados en las páginas. Estas plantillas son utilizadas por Struts y los procesos de importación de datos. |
| **struts.xml** | Es el fichero de configuración de Struts.  Ver también: **presentación.xml**. |
| **tiles.xml** | Es el fichero de configuración de Tiles. Contiene la declaración de las plantillas que se usan para construir las páginas de la aplicación. |
| **pom.xml** | Este fichero es utilizado por Maven. Contiene la declaración de todas las dependencias (librerías), scripts para generar stubs de los servicios web, etc. Ver más detalles en: **notas.html** en el proyecto Eclipse. |
| **\*.xlsx** | Son las plantillas que se usan en los procesos de exportación e importación de operaciones (carga masiva). |
| **\*.wsdl** | Es la definición de un servicio web consumido por la aplicación (Ej: @Firma). Estos ficheros son utilizados por Maven para generar las clases de los stubs en Java. |
|  |  |
|  |  |

Parámetros de **gesfse.properties**

|  |  |
| --- | --- |
| Nombre del fichero | Descripción |
| **mecd.url** | Es la URL del dominio de despliegue del servidor. Por ejemplo: <https://aplipre.mecd.es>. Esta URL se utiliza en los correos de activación de las cuentas de los usuarios para construir el enlace que debe pinchar el usuario. El valor es distinto para cada entorno de despliegue |
| **servicioVerificacion.endpoint** | Es el endpoint del servicio web de @Firma. Se invoca para validar los certificados de los usuarios que se identifican con un certificado digital |
| **documentum.endpoint** | Es el endpoint del servicio web intermediario de Documentum |
| **documentum.atom.url** | La URL para acceder a Documentum |
| **documentum.usuario** | ID del usuario para conectarse con Documentum |
| **documentum.contrasena** | Contraseña para conectarse con Documentum |
| **email.activo** | Un flag que indica si se pueden enviar correos. Los valores posibles son: **false** y **true**. |
| **email.remitente** | Cuenta de correo del remitente para los correos enviados por la aplicación |
|  |  |

# Convenciones del Desarrollo

## Estructura del Proyecto

La aplicación se diseña siguiendo la definición del estándar J2EE. La estructura del proyecto en Eclipse está creada con el arquetipo de aplicaciones web de Maven, que se muestra a continuación.



## Definición de los Paquetes de la Aplicación

El proyecto (Eclipse) tiene por un lado, las clases y recursos que corresponden a la implementación de la aplicación (**src/main**) y por otro las clases y recursos de las pruebas unitarias (**src/test**). El fichero **pom.xml** contiene la definición de las dependencias, targets y demás parámetros utilizados por Maven.

Siguiendo las convenciones ya definidas en MECD, los paquetes del proyecto están organizados de la siguiente manera:



## Nombres de Clases, Métodos y Variables

En general, se adoptan todos los estándares y convenciones definidos para el lenguaje Java. Los nombres de las clases, métodos y variables deben ser siempre castellanos o castellanizados, evitando caracteres especiales como ñ, tildes, etc. Se admite el uso de prefijos o sufijos en inglés a efectos de *homogeneizar* la nomenclatura de clases, propiedades y métodos para adecuarlos a la tecnología específica que se ha decidido utilizar. Por ejemplo: usar “**get**” ,“**set**” o “**is**” para los *beans*.

Los nombres de las clases corresponden a *sustantivos* y son descriptivos de la función dentro del modelo de objetos de la aplicación. En algunos casos también se añade un sufijo que especifica su papel dentro de la jerarquía de capas. Por ejemplo: **OperacionEntity**, **OperacionRepository**, etc.

Los nombres de los métodos deben ser siempre *verbos*. Si se trata de nombres compuestos por varias palabras, la primera letra de cada palabra es una mayúscula. Por ejemplo: **setNombre(…)**, **procesarOperacionesPendientes()**, etc.

Los nombres de las variables de instancia y variables locales no pueden comenzar con “**\_**” o “**$**”. Deben ser breves y comunicar el significado de forma evidente. Se evitarán las variables de un carácter, con la sola excepción de las variables de las cláusulas **catch()** y los contadores temporales de los bucles (**e**, **i**, **j**, **k**, **m**). Los nombres de las variables no tendrán ningún prefijo que indique el tipo del objeto u otro clasificador. Los nombres de constantes van en mayúsculas y con un “**\_**” para separar las palabras (por ejemplo: **CANTIDAD\_MAXIMA**).

## Gestión de Excepciones

Las excepciones son errores que ocurren durante la ejecución de la aplicación y afectan el flujo normal de funcionamiento. Java ofrece una infraestructura robusta y basada en objetos para gestionar los diferentes escenarios de excepciones.

En general, cuando se produce una excepción, la aplicación debe tratarla, propagarla o generar una traza en el fichero de log. Si el bloque de código donde ocurre la excepción puede tratarla, entonces deberá hacerlo. Caso contrario, por normal general, las excepciones serán propagadas y tratadas en los niveles superiores.

Nunca deben usarse cláusulas try/catch vacías y siempre debe dejarse una traza en el fichero de log con toda la pila de ejecución correspondiente. Las trazas serán generadas con la librería Log4J y el fichero de configuración irá embarcado dentro de la misma aplicación (**.war**). En ningún caso se deben generar trazas con **System.out** o **System.err**.

Las clases de excepciones específicas de la aplicación seguirán las convenciones definidas previamente para el nombrado de clases y además deben terminar con el sufijo “**Exception**”. Por ejemplo: “**ValidacionUsuarioException**”.

Finalmente, debe tenerse en cuenta que las excepciones son caras de gestionar y que deben usarse razonablemente. Esto significa que debe analizarse si en algunos casos es más simple retornar un *flag* booleano para indicar que una operación se realizó con éxito o no—en lugar de lanzar y propagar una excepción.

## Plantillas y Ensamblado de las páginas

Las páginas HTML que representan la funcionalidad y navegación de la aplicación para los usuarios finales serán generadas a partir de una o varias plantillas. Así, cada página únicamente cambia una porción—normalmente el cuerpo central—del contenido mostrado al usuario.

Esta forma de trabajo permite identificar y definir fragmentos que pueden ser ensamblados dinámicamente en tiempo de ejecución para crear las páginas HTML. El uso de fragmentos permite reducir la duplicación de elementos comunes a todas las páginas y trabajar con plantillas que pueden reutilizarse.

Más concretamente, las páginas de la aplicación incluirán los siguientes elementos: una cabecera de común, un pie, un menú de opciones y un cuerpo.

Las plantillas serán creadas con la librería Tiles y serán integradas en la capa de presentación con Struts. La construcción de páginas a partir de plantillas se basa en la construcción de tres elementos:

* La plantilla que define elementos estáticos y áreas donde insertar otros contenidos
* La página real que utiliza la plantilla insertando dinámicamente textos y otros elementos
* Las porciones reutilizables, que pueden ser otras páginas JSP

## Convenciones para los Objetos de la Base de Datos

A continuación se mencionan las principales convenciones que se adoptan en la creación de los objetos de la base de datos y que se basan en la documentación aportada por el MECD.

En general, todos los nombres constarán de una sola palabra, pero se admite el símbolo “**\_**” para unir las palabras en nombres compuestos. Todos los nombres se escribirán en castellano o serán castellanizados y utilizarán caracteres alfabéticos o dígitos (0-9). No se utilizarán vocales acentuadas ni caracteres propios de un idioma (como ‘ñ’, apóstrofes...) ni artículos, conjunciones, ni partículas de unión.

Para las tablas y vistas deben utilizarse nombres en singular. En el caso de tablas o vistas que procedan de una relación **M:N** entre dos tablas padre, el nombre será compuesto de la denominación de las tablas padre, intentando dar preferencia a la entidad padre más relevante o con más peso en la relación. En el caso de una vista que proceda de una tabla-base, se nombrará a partir de la tabla de origen, precedida del prefijo “**V\_**”. Se utilizarán numerales en caso necesario. Por ejemplo: “**V\_Persona1**”, “**V\_Persona2**”.

Si una vista se ha creado para su uso específico desde un esquema concreto, se colocará como sufijo el identificador de dicho esquema. Ejemplo: si se crea una vista a una tabla '**Persona**' para su uso desde un esquema llamado CTLXXADM, se nombraría como “**V\_PERSONA\_CTL**”

En el caso de entidades auxiliares que se encuentren en otro esquema (como tablas de propósito general), se procede de la siguiente manera:

* Si se utilizan la mayor parte de los atributos y tuplas de la entidad auxiliar, se referenciará ésta mediante un sinónimo, el cual mantendrá el nombre de la entidad original (aunque se encuentre en plural)
* Si se utiliza un subconjunto reducido de los atributos y tuplas de la entidad auxiliar, se accederá a ella mediante una vista. En ese caso, el nombre seguirá los mismos convenios que para tablas.

Los nombres de las secuencias comienzan siempre por “**S\_**”. Si las utiliza una sola tabla, llevarán a continuación el nombre de ésta. Si las utilizan varios objetos o tablas, tendrán un nombre compuesto “**\_**”.

Los nombres de los procedimientos almacenados comienzan por P\_. La siguiente palabra debe ser un verbo infinitivo indicativo de su propósito principal (Ejemplo: “**P\_ACTUALIZAR**...”, “**P\_CALCULAR...**”). Los nombres de las funciones comienzan por “**F\_**”. Como antes, la siguiente palabra será un verbo infinitivo indicativo de su propósito principal.

Los paquetes tienen un formato libre, pero sus procedimientos y funciones componentes respetarán las normas anteriores.

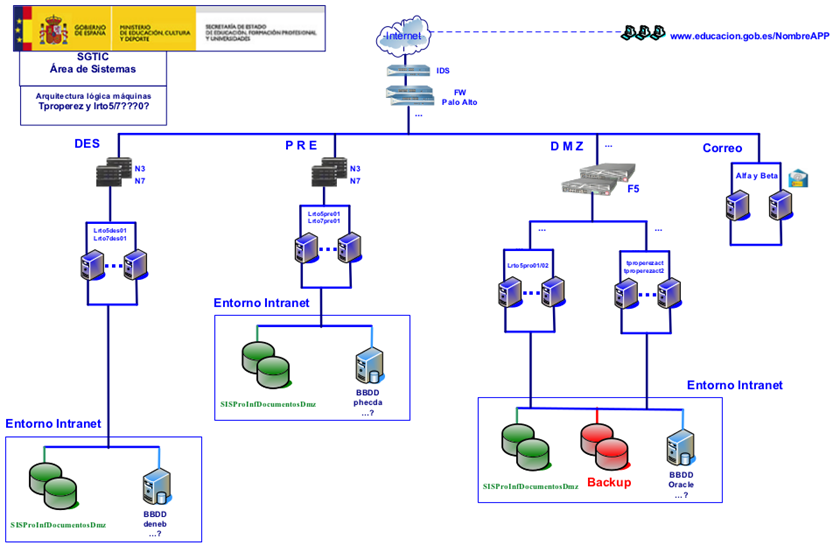
Las claves primarias deben nombrarse como la tabla y añadiendo el sufijo “**\_PK**”. Los índices únicos también se nombran como la tabla y añadiendo el sufijo “**\_U99**”, donde ‘**99**’ son dos posiciones para numeral correlativo. Los índices no únicos se nombran como la tabla y añadiendo el sufijo “**\_I99**”, donde ‘**99**’ son dos posiciones para numeral correlativo.

Las claves ajenas se nombran como la tabla en la que se define la restricción de clave ajena y se añade el sufijo “**\_R99**”, donde ‘**99**’ son dos posiciones para numeral correlativo.

# Arquitectura Física y Despliegue

## Topología de la Plataforma

El siguiente diagrama muestra la configuración de los diferentes entornos del MECD para el despliegue de las aplicaciones.



Topología de los entornos

Según la documentación aportada por el MECD que describe la plataforma tecnológica de su CPD, hay que destacar los entornos de ejecución sobre los se basa el desarrollo e implementación del sistema demandado:

* Sistema operativo: Red Hat Linux 6.0, bajo Hypervisor VMware 4.x
* Servidor de aplicaciones para los despliegues de las aplicaciones y servicios: Tomcat 7.x
* Servicios de identidades bajo LDAP tecnología Microsoft o SSO desarrollo ad-hoc MECD
* Base de datos Oracle 11g
* Gestor Documental: Documentum
* Gestión de datos con Visual Analytics
* Suite soluciones comunes: red SARA, @Firma y demás servicios incluidos en el Catálogo de Servicios Comunes de la DTIC

## Despliegue en los entornos de Pre-Producción y Producción

Como parte del ciclo de vida de los desarrollos, éstos deben recorrer diferentes entornos enfocados a la consecución de diferentes objetivos. Por ello una vez terminada la fase de desarrollo es necesario realizar diferentes tareas, como pruebas de rendimiento o aceptación. Para ello el MECD cuenta con un entorno específico de Pre-Producción (también denominado entorno de Calidad), como se mencionaba en el apartado anterior.

Dada la importancia del paso de una aplicación a los entornos Pre-Producción y Producción, el MECD cuenta con un procedimiento específico para el manejo de este cambio. Los despliegues en dichos entornos serán realizados exclusivamente por personas del MECD. El equipo de desarrollo podrá realizar validaciones de los despliegues y consultas de los ficheros de log con las trazas de ejecución.

# Entorno de Desarrollo

## Herramientas

El entorno de desarrollo tiene dos caras: por un lado, están los ordenadores individuales con las herramientas utilizadas por los desarrolladores y por otro están las herramientas compartidas a través de la intranet.



Herramientas del entorno de desarrollo (puesto local)

Cada puesto de desarrollo cuenta con una instalación completa de las siguientes herramientas: JDK, Eclipse, Tomcat, Maven, SQLDeveloper y Notedpad++. La instalación de Eclipse incluye varios *plug-ins* que permiten acceder a SVN, extraer métricas calidad del código fuente e integrar el proyecto con Maven.

El entorno de desarrollo también incluye herramientas compartidas por la intranet por todo el equipo: el repositorio de código fuente (SVN), bases de datos y un servidor de integración continua (IC). Nótese que en el diagrama, el icono de la base de datos representa el esquema con las tablas de las aplicaciones propias del proyecto y también las *tablas maestras* comunes a otras aplicaciones (códigos de países, códigos de CCAA, etc.).



Herramientas del entorno de desarrollo compartidas (intranet)

A continuación se describen brevemente las principales herramientas mencionadas en el apartado anterior:

**Maven** es una herramienta open-source que simplifica los procesos de construcción y gestión de los ficheros de instalación de las aplicaciones y servicios (**.war**), a partir del código fuente. Entre las principales características de Maven, destacan las siguientes:

* Estandarización de la estructura interna de los proyecto (carpetas de código fuente, ubicación de ficheros)
* Simplificación de la gestión de las librerías comunes (esto se denomina “*gestión de dependencias*”). Las librerías se descargan desde *repositorios públicos* accesibles por Internet.
* Automatización de tareas rutinarias, como la compilación y generación de ficheros de despliegue (**.war**)
* Cada proyecto tiene un fichero *pom.xml* con la configuración utilizada por Maven
* Se invoca por línea de comandos o desde el menú de Eclipse

**Subversion (SVN)** es una herramienta de control de versiones del código fuente que se basa en el uso de un repositorio central—accesible por la intranet—cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros. Con SVN, los cambios producidos en el repositorio se denominan *revisiones*.

En nuestros proyectos, accederemos al repositorio central desde un *plug-in* de Eclipse. Desde allí los desarrolladores harán todas las operaciones de *check-in*, *check-out* y etiquetado del código fuente.

## Versiones de las Herramientas

|  |  |  |
| --- | --- | --- |
| Herramienta | Versión | Descripción |
| Java Development Kit (JDK) | 1.7 | Es la máquina virtual de Java y el compilador. Distribución libre y gratuita |
| Eclipse | Luna | Es el IDE de desarrollo. Distribución libre y gratuita |
| Plug-ins de Eclipse: SVN, Maven, PMD, Findbugs, CheckStyle | Compatible con Eclipse | Son extensiones de Eclipse (plug-ins) que se usan para conectarse a SVN, Maven y sacar métricas de calidad del código fuente. Distribución libre y gratuita |
| Apache Tomcat | 7 | Es el servidor de despliegue de las aplicaciones. Distribución libre y gratuita |
| SQLDeveloper | 4 | Herramienta para consultar una base de datos Oracle. Distribución libre y gratuita |
| Apache Maven | 3.x | Automatiza la gestión de las librerías utilizadas por las aplicaciones (dependencias). Distribución libre y gratuita |
| Notepad++ | 6.x | Editor de textos (XML, HTML, ficheros de configuración, etc.). Distribución libre y gratuita |
| Firefox | 40 | Navegador de internet. Distribución libre y gratuita |
| Internet Explorer | 9 | Navegador de internet. Distribución libre y gratuita |
| Microsoft Word | 2010 | Editor de textos para la documentación |

## Ciclo de Desarrollo con SVN y Maven

El siguiente diagrama resume el ciclo de desarrollo de las aplicaciones y servicios utilizando las herramientas mencionadas previamente:

* Cada desarrollador trabaja desde su ordenador y accede al repositorio central de Subversion (SVN) para hacer *check-in* y *check-out* del código fuente
* Las librerías utilizadas por los proyectos son gestionadas automáticamente por Maven, que se encarga de descargarlas bajo demanda desde los repositorios públicos



Ciclo de desarrollo en el entorno compartido