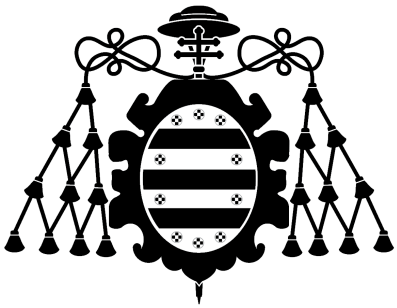


UNIVERSIDAD DE OVIEDO

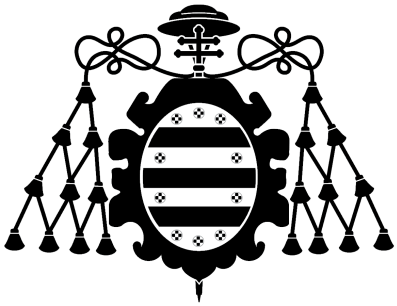
ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

SPRING ROO ADD-ONS PARA PROTOTIPADO RÁPIDO

JAVIER MENÉNDEZ ÁLVAREZ
JULIO 2014



UNIVERSIDAD DE OVIEDO

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

MÁSTER EN INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

SPRING ROO ADD-ONS PARA PROTOTIPADO RÁPIDO

DOCUMENTO Nº V

DISEÑO DEL SISTEMA DE INFORMACIÓN

JAVIER MENÉNDEZ ÁLVAREZ
JULIO 2014

ÁREA DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL
TUTOR: Mª JOSÉ SUÁREZ CABAL

Índice

Introducción

Este documento contiene el Diseño del Sistema de Información (DSI) del add-on para Spring Roo, Roo++.

Se describirá cómo debe realizarse la aplicación, mediante el diseño de la arquitectura del sistema, el diseño detallado de los subsistemas, y el diseño de datos.

El diseño de la arquitectura del sistema incluye la descripción de los requisitos tecnológicos de implantación y desarrollo de este software, y el diagrama de paquetes, que refleja los subsistemas que componen la aplicación y sus dependencias.

El diseño detallado de los subsistemas permitirá conocer los métodos que conforman cada uno de los subsistemas documentando las interfaces externas del propio subsistema, la estructura interna a nivel de unidades y el detalle de los módulos más complejos.

El diseño de datos consistirá fundamentalmente en el diseño de la estructura de ficheros y directorios que la aplicación va a manejar, tanto los ficheros de configuración en los que el usuario ha de establecer el comportamiento del software como los ficheros que forman parte de cada proyecto desarrollado con Roo++, además de los ficheros que se encuentran contenidos en "resources" dentro del JAR de Roo++.

Diseño de la arquitectura

1. Entornos tecnológicos

Roo++ está basado en el uso de las siguientes tecnologías:

- **Java:** plataforma de software (tanto desarrollo como ejecución) y lenguaje utilizado en la misma, creados por la compañía Sun Microsystems, posteriormente adquirida por Oracle. Las tecnologías de Java (lenguaje de programación, entorno de ejecución, conocido como Java Runtime Environment (JRE), y plataforma de desarrollo, conocida como Java Development Kit (JDK)) son utilizadas tanto en el desarrollo como en el uso de Roo++, siendo necesaria la versión Java 6 o posterior. Además, en el tipo de desarrollos para los que está diseñado Roo++ es de gran utilidad el uso de Java EE (Enterprise Edition) en lugar de Java SE (Standard Edition).
- **Apache Maven:** herramienta de automatización de la construcción (compilación) de proyectos, con un propósito parecido al de otra herramienta similar, Apache Ant, si bien con un funcionamiento diferente. Un proyecto Maven puede componerse de uno o varios módulos (éste último será el caso de los proyectos desarrollados con Roo++). Cada uno de estos módulos contiene un archivo XML llamado *pom.xml* (de *Project Object Model*), en el que se definen aspectos como las dependencias con otros módulos y componentes (indicando siempre 3 campos: *groupId, artifactId* y *version*), o los plug-ins requeridos. También existe un archivo *pom.xml* adicional para el proyecto completo, siempre que éste se componga de varios módulos, en el cual, además, se indica el orden en que se han de compilar estos módulos. Todo proyecto desarrollado con Roo está preparado para ser compilado directamente con Maven. Además, éste último también es utilizado para la compilación de Roo++.
- **Spring Roo:** sistema de desarrollo rápido de aplicaciones empresariales basadas en Java, desarrollado por SpringSource, una división de la empresa VMware. Hace uso de otras tecnologías, como Spring Framework (también desarrollado por SpringSource), JavaServer Pages, Java Persistence API, o Apache Maven. Una de sus principales ventajas es que no tiene ningún componente específico requerido en tiempo de ejecución de las aplicaciones desarrolladas, pudiendo incluso ser eliminado por completo de un proyecto ya desarrollado en tan solo unos minutos. Roo proporciona una serie de comandos para llevar a cabo el desarrollo de las aplicaciones, los cuales son proporcionados por una serie de "base add-ons" que vienen incluídos en la instalación de Roo. Roo++ proporciona un add-on adicional que incluye nuevos comandos para facilitar aún más el desarrollo, si bien restringiendo algo el ámbito de las aplicaciones a desarrollar. Esta versión de Roo++ ha sido desarrollada para Spring Roo 1.2.3.
- **Spring Tool Suite:** entorno integrado de desarrollo (IDE), basado en el bien conocido Eclipse, y optimizado para el desarrollo de aplicaciones mediante Spring Roo. Con él se ha desarrollado Roo++, y también es el entorno más apropiado para desarrollar los proyectos en que éste se utiliza. Alternativamente, el plug-in de Spring Tool Suite ofrece una funcionalidad similar sobre una instalación de Eclipse ya existente. Al igual que Roo, Spring Tool Suite ha sido desarrollado por SpringSource.
- **Servidores web:** las aplicaciones web desarrolladas con Roo++ pueden ser desplegadas en los servidores web Tomcat y Jetty. Además, también es posbile su despliegue en servidores de aplicaciones, tales como JBoss.

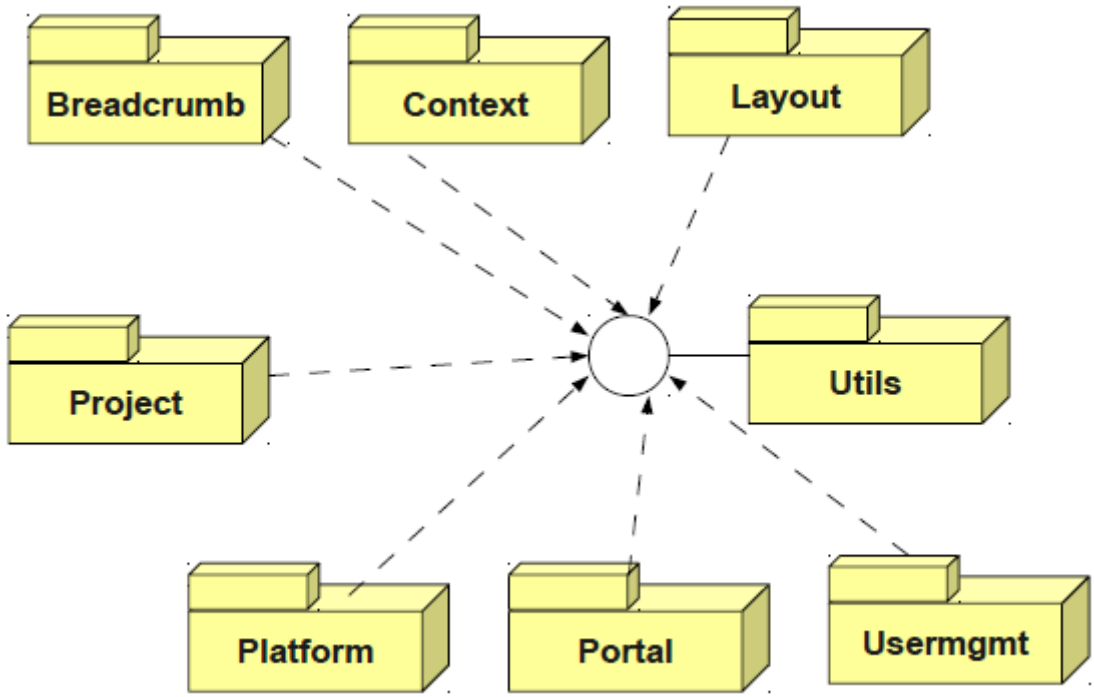
- **Sistemas de gestión de bases de datos:** las aplicaciones desarrolladas con Roo++ pueden hacer uso de bases de datos de Oracle, MySQL, PostgreSQL y Microsoft SQL Server, que son, en general, los sistemas de gestión de bases de datos más utilizados en este tipo de aplicaciones.
- **OSGi:** sistema que permite la introducción, actualización o eliminación de módulos de forma dinámica. Es utilizado por Spring Roo para la gestión de los add-ons, tanto los que forman parte de su instalación como de add-ons adicionales (por ejemplo, Roo++).

Roo++ no tiene ningún requisito especial adicional para su desarrollo o implantación; cualquier hardware y sistema operativo que pueda hacer uso de todas estas tecnologías puede hacerlo de Roo++.

En cuanto al desarrollo, Roo++ ha sido desarrollado con Java, Spring Tool Suite, Apache Maven y OSGi, además de Spring Roo para las pruebas y depuración (no así para el desarrollo propiamente dicho; aunque Roo++ es un add-on de Spring Roo, no ha sido desarrollado haciendo uso de éste).

2. Especificación de los subsistemas

A continuación se muestran los distintos subsistemas de los que se compone el software, con las relaciones de dependencia que existen entre ellos, y, finalmente, una descripción general del cometido de cada uno de ellos.



Subsistema	Descripción
Breadcrumb	Determina la disponibilidad, y lleva a cabo la ejecución, del comando `roo++ web breadcrumb setup`, que hace posible la navegación mediante "migas de pan" (*Anterior*, *Siguiente*, *Inicio*, etc) en la aplicación web desarrollada.
Context	Determina la disponibilidad, y lleva a cabo la ejecución, del comando `roo++`

	`context setup`, que hace posible una separación lógica de los datos de la aplicación, mostrando un conjunto de datos diferente a cada usuario de la aplicación web desarrollada según cuál sea su perfil.
Layout	Determina la disponibilidad, y lleva a cabo la ejecución, del comando `roo++ layout setup`, que permite realizar cambios en la apariencia de la interfaz web de la aplicación desarrollada.
Platform	Determina la disponibilidad, y lleva a cabo la ejecución, de los comandos `roo++ platform setup` y `roo++ platform configuration`, que hacen posible que el proyecto en desarrollo haga uso de la plataforma software de la organización que lo está desarrollando.
Portal	Determina la disponibilidad, y lleva a cabo la ejecución, del comando `roo++ portal setup`, que permite integrar el proyecto en desarrollo en el portal web de la organización a que éste pertenece.
Project	Determina la disponibilidad, y lleva a cabo la ejecución, de los comandos `roo++ project create-modules`, `roo++ project license`, `roo++ project web configuration`, `roo++ project web jpa`, y `roo++ project deploy`, que permiten llevar a cabo una serie de tareas relacionadas con el proyecto en desarrollo y con su aplicación web.
Usermgmt	Determina la disponibilidad, y lleva a cabo la ejecución, del comando `roo++ usermgmt setup`, que introduce la autenticación de usuarios en la aplicación en desarrollo.
Utils	Implementa cualquier tarea compleja o repetitiva que se ha de llevar a cabo, desde algunas específicas para ser usadas desde un subsistema concreto (por ejemplo, “configurar la gestión de usuarios”, para ser usada siempre desde el subsistema *Usermgmt*) hasta otras más genéricas, que pueden ser usadas desde cualquier subsistema que las necesite (por ejemplo, “copiar fichero”).

Diseño detallado de los subsistemas

A continuación se describirá en detalle cada uno de los subsistemas que componen el add-on para Spring Roo, Roo++. Se describirá cada uno de los subsistemas, así como cada una de las clases Java que los componen, y cada una de las interfaces (tanto internas como externas) que proporcionan.

Excepto el subsistema *Utils,* que contiene diversas clases de utilidades, todos los demás subsistemas se encargan de 2 tareas: determinar si uno o varios comandos están disponibles en este momento, y realizar la tarea correspondiente a la ejecución de ese o esos mismos comandos (generalmente, invocando a interfaces externas del subsistema *Utils*).

Por ello, la implementación de todos los subsistemas, exceptuando *Utils*, consistirá en 4 unidades (clases Java):

1. Clase que contiene, para cada comando de Roo++ perteneciente al subsistema, 2 módulos (métodos de la clase Java), uno con las anotacións que indica que el método determina la disponibilidad del comando para Roo, y otro con la anotación que indica que ese método es lo que ha de ejecutarse cuando se introduce en Roo la orden y los parámetros indicados. Todos estos métodos serán invocados por Roo de forma automática en el momento que sea necesario su uso, gracias a las anotaciones que así se lo indican.
2. Clase abstracta que define los métodos que llevan a cabo, para cada uno de los comandos Roo++ del subsistema, las 2 operaciones anteriormente mencionadas, y ocasionalmente alguna funcionalidad adicional.
3. Implementación de la clase abstracta anterior.
4. Declaración del paquete Java al que pertenecen las 4 clases del subsistema (esta declaración del paquete también está presente en el subsistema *Utils*).

1. Breadcrumb

Subsistema que permite introducir la navegación mediante “migas de pan” (es decir, enlaces a la página inicial, o a la anterior y a la siguiente, etc.) en la aplicación en desarrollo, mediante el uso de la plataforma software de la organización, haciendo uso de los beans que se encarguen de ello.

Las interfaces de la clase encargada de llevar a cabo la determinación de la disponibilidad y la ejecución de comandos son externas, y están destinadas a ser usadas por Roo, según las anotaciones Java que así se lo indican, mientras que las de las clases de operaciones son internas, destinadas a ser usadas desde la primera.

1.1 BreadcrumbCommands.java

Comandos relacionados con las "migas de pan" del módulo web de los proyectos.

@CliAvailabilityIndicator("roo++ web breadcrumb setup") **public boolean isBreadcrumbSetupAvailable();**
Indica si es posible ejecutar el comando web breadcrumb setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ web breadcrumb setup", help = "Sets up breadcrumbs for

web application") **public void breadcrumbSetup();**
Configura las "migas de pan" o breadcrumb para el módulo web del proyecto que tiene el foco.
Devuelve: nada.

1.2 BreadcrumbOperations.java

Operaciones relacionadas con las "migas de pan" del módulo web de los proyectos.

boolean canBreadcrumbSetup();
Indica si es posible ejecutar el comando web breadcrumb setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

void breadcrumbSetup();
Configura las "migas de pan" o breadcrumb para el módulo web del proyecto que tiene el foco.
Devuelve: nada.

1.3 BreadcrumbOperationsImpl.java

Implementación de los métodos declarados en *BreadcrumbOperations.java*.

1.4 package-info.java

Declaración del paquete para la configuracion de las "breadcrumb" o "migas de pan" de la interfaz web de una aplicacion Spring Roo.

2. Context

Subsistema que, a través de la plataforma software de la organización, permite definir diferentes subconjuntos de datos, de entre el total de datos que maneja la aplicación desarrollada, de forma que cada usuario, según su perfil y sus permisos, acceda a uno de ellos o a otro.

Las interfaces de la clase encargada de llevar a cabo la determinación de la disponibilidad y la ejecución de comandos son externas, y están destinadas a ser usadas por Roo, según las anotaciones Java que así se lo indican, mientras que las de las clases de operaciones son internas, destinadas a ser usadas desde la primera.

2.1 ContextCommands.java

Comandos relacionados con el establecimiento de contextos (separación lógica de los datos de la aplicación, a los cuales cada usuario podrá acceder según sus permisos).

@CliAvailabilityIndicator("roo++ context setup") **public boolean isContextSetupAvailable();**

Indica si es posible ejecutar el comando context setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ context setup", help = "Configures context for user management") **public void contextSetup();**
Crea contextos (separación lógica de los datos de la aplicación, a los cuales cada usuario podrá acceder segun sus permisos).
Devuelve: nada.

2.2 ContextOperations.java

Operaciones relacionadas con el establecimiento de contextos (separación lógica de los datos de la aplicación, a los cuales cada usuario podrá acceder según sus permisos).

boolean canContextSetup();
Indica si es posible ejecutar el comando context setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

void contextSetup();
Crea contextos (separación lógica de los datos de la aplicación, a los cuales cada usuario podrá acceder según sus permisos).
Devuelve: nada.

2.3 ContextOperationsImpl.java

Implementación de los métodos declarados en *ContextOperations.java*.

2.4 package-info.java

Declaración del paquete de establecimiento de contextos.

3. Layout

Subsistema que permite introducir nuevos elementos relacionados con la visualización de la aplicación web, tales como imágenes, layouts, hojas de estilos, JavaScript, etc.

Las interfaces de la clase encargada de llevar a cabo la determinación de la disponibilidad y la ejecución de comandos son externas, y están destinadas a ser usadas por Roo, según las anotaciones Java que así se lo indican, mientras que las de las clases de operaciones son internas, destinadas a ser usadas desde la primera.

3.1 LayoutCommands.java

Comandos relacionados con el layout de la aplicación del módulo web de los proyectos.

@CliAvailabilityIndicator("roo++ layout setup") **public boolean isLayoutSetupAvailable();**
Indica si es posible ejecutar el comando layout setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ layout setup", help = "Configures web app layout") **public void layoutSetup();**
Configura el layout (apariencia, estilo) del módulo web del proyecto que tiene el foco.
Devuelve: nada.

3.2 LayoutOperations.java

Operaciones relacionadas con el layout de la aplicación del módulo web de los proyectos.

boolean canLayoutSetup();
Indica si es posible ejecutar el comando layout setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

void layoutSetup();
Configura el layout (apariencia, estilo) del módulo web del proyecto que tiene el foco.
Devuelve: nada.

3.3 LayoutOperationsImpl.java

Implementación de los métodos declarados en *LayoutOperations.java*.

3.4 package-info.java

Declaración del paquete para la configuración de la apariencia de la interfaz web de una aplicación Spring Roo.

4. Platform

Subsistema que hace posible el uso de la plataforma software de la organización desde la aplicación que está siendo desarrollada.

Las interfaces de la clase encargada de llevar a cabo la determinación de la disponibilidad y la ejecución de comandos son externas, y están destinadas a ser usadas por Roo, según las anotaciones Java que así se lo indican, mientras que las de las clases de operaciones son internas, destinadas a ser usadas desde la primera.

4.1 PlatformCommands.java

Comandos relacionados con el uso de una plataforma (externa a Roo++) en los proyectos.

@CliAvailabilityIndicator("roo++ platform setup") **public boolean isPlatformSetupAvailable();**
Indica si es posible ejecutar el comando platform setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ platform setup", help = "Adds platform dependencies to project") **public void platformSetup();**
Introduce en el proyecto que tiene el foco dependencias respecto a una plataforma configurada por el usuario.
Devuelve: nada.

@CliAvailabilityIndicator("roo++ platform configuration") **public boolean isPlatformConfigurationAvailable();**
Indica si es posible ejecutar el comando platform configuration en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ platform configuration", help = "Adds platform configuration beans to web module") **public void platformConfiguration();**
Introduce en el módulo web del proyecto que tiene el foco los beans necesarios para el uso de la plataforma anteriormente configurada.
Devuelve: nada.

4.2 PlatformOperations.java

boolean canPlatformSetup();
Determina si es posible ejecutar el comando platform setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario

void platformSetup();
Introduce en el proyecto que tiene el foco dependencias respecto a una plataforma configurada por el usuario.

Devuelve: nada.

boolean isPlatformSetup();
Determina si ya se han introducido en el proyecto que tiene el foco dependencias con respecto a una plataforma.
Devuelve: true si ya se han introducido las dependencias de la plataforma; false en caso contrario.

boolean isPlatformConfigured();
Determina si ya se han introducido en el módulo web del proyecto que tiene el foco los beans necesarios para el uso de la plataforma anteriormente seleccionada.
Devuelve: true si ya se han introducido los beans de la plataforma; false en caso contrario.

void platformConfiguration(String internalPath);
Introduce en el módulo web del proyecto que tiene el foco los beans necesarios para el uso de la plataforma anteriormente seleccionada.
internalPath: ruta interna, dentro del directorio "resources" del JAR, del directorio donde se encuentran realmente todos los resources.
Devuelve: nada.

4.3 PlatformOperationsImpl.java

Implementación de los métodos declarados en *PlatformOperations.java*.

4.4 package-info.java

Declaración del paquete para la configuración del uso de una plataforma externa desde una aplicación Spring Roo.

5. Portal

Subsistema que permite integrar la aplicación desarrollada en el portal web de la organización, a través del uso de los beans de la plataforma software que ésta posee.

Las interfaces de la clase encargada de llevar a cabo la determinación de la disponibilidad y la ejecución de comandos son externas, y están destinadas a ser usadas por Roo, según las anotaciones Java que así se lo indican, mientras que las de las clases de operaciones son internas, destinadas a ser usadas desde la primera.

5.1 PortalCommands.java

Comandos relacionados con el uso de un portal en el módulo web de los proyectos.

@CliAvailabilityIndicator("roo++ portal setup")

public boolean isPortalSetupAvailable();
Indica si es posible ejecutar el comando portal setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ portal setup", help = "Configures web module to use organization's portal") **public void portalSetup();**
Configura el módulo web del proyecto que tiene el foco para el uso del portal de la organización, según lo establecido por el usuario.
Devuelve: nada.

5.2 PortalOperations.java

void portalSetup(String internalPath);
Configura el módulo web del proyecto que tiene el foco para el uso del portal de la organización, según lo establecido por el usuario.
internalPath: ruta interna, dentro del directorio "resources" del JAR, del directorio donde se encuentran realmente todos los resources.
Devuelve: nada.

boolean isPortalSetupAvailable();
Indica si es posible ejecutar el comando portal setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

5.3 PortalOperationsImpl.java

Implementación de los métodos declarados en *PortalOperations.java*.

5.4 package-info.java

Declaración del paquete para la configuración de un portal para el módulo web de una aplicación Spring Roo.

6. Project

Subsistema que permite realizar múltiples operaciones relacionadas con aspectos generales de los proyectos: crear sus módulos y adaptarlos al uso de Roo++, introducir una o varias licencias, prepararlos para el despliegue en los servidores web Tomcat y Jetty de forma que se puedan realizar pruebas durante el desarrollo, o definir la conexión con una base de datos.

Las interfaces de la clase encargada de llevar a cabo la determinación de la disponibilidad y la ejecución de comandos son externas, y están destinadas a ser usadas por Roo, según las anotaciones

Java que así se lo indican, mientras que las de las clases de operaciones son internas, destinadas a ser usadas desde la primera.

6.1 ProjectCommands.java

Comandos relacionados con la creación y configuración de proyectos.

@CliAvailabilityIndicator("roo++ project create-modules") **public boolean isCreateModulesAvailable();**
Indica si es posible ejecutar el comando project create-modules en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ project create-modules", help = "Creates the modules for a roo++ project") **public void createModules(@CliOption(key = "version", help = "Project version number", unspecifiedDefaultValue = "1.0.BUILD-SNAPSHOT") String version);**
Crea los diferentes módulos del proyecto Maven/Spring Roo que tiene el foco.
version: nombre de la versión que el usuario asigna al proyecto para el que se estan creando los módulos.
Devuelve: nada.

@CliAvailabilityIndicator("roo++ project license") **public boolean isLicenseAvailable() ;**
Indica si es posible ejecutar el comando project license en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ project license", help = "Configures the license for an roo++ project") **public void license();**
Configura la licencia del proyecto que tiene el foco.
Devuelve: nada.

@CliAvailabilityIndicator("roo++ project web configuration") **public boolean isWebConfigurationAvailable();**
Indica si es posible ejecutar el comando project web configuration en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ project web configuration", help = "Configures the web module for an roo++ project") **public void webConfiguration();**
Realiza diferentes acciones de configuración sobre el módulo web del proyecto que tiene el foco.
Devuelve: nada.

@CliAvailabilityIndicator("roo++ project web jpa") **public boolean isWebJpaAvailable();**
Indica si es posible ejecutar el comando project web jpa en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ project web jpa", help = "Configures the web module persistence with JPA") **public void webJpa(@CliOption(key = "user", mandatory = true, help = "Database user name") String user,** **@CliOption(key = "password", mandatory = true, help = "Database password") String password,** **@CliOption(key = "database", help = "Database name (project name if not specified); it will not be used with Oracle, as database name is the same as user name", unspecifiedDefaultValue = "") String database,** **@CliOption(key = "dbms", help = "Database management system to be used with the project: oracle (default), mysql, mssql, postgres", unspecifiedDefaultValue = "oracle") String dbms,** **@CliOption(key = "server", help = "Machine who acts as database server", unspecifiedDefaultValue = "localhost") String server,** **@CliOption(key = "port", help = "Port used for database communications (default installation port for selected DBMS if not specified)", unspecifiedDefaultValue = "") String portNumber,** **@CliOption(key = "hibernateversion", help = "Version of Hibernate to be used", unspecifiedDefaultValue = "") String hibernateVersion);**
Configura el módulo "web" del proyecto que tiene el foco para el uso de una base de datos.
user: nombre de usuario para el acceso a la base de datos.
password: contraseña para el acceso a la base de datos.
database: nombre de la base de datos (que será el nombre del proyecto si no se indica).
dbms: nombre del sistema de gestión de bases de datos: oracle (valor por defecto), mysql, mssql, postgres.
server: máquina que contiene la base de datos.
portNumber: puerto a través del cual se accede a la base de datos.

hibernateVersion: versión de Hibernate a utilizar. Este parámetro no estaba previsto inicialmente; debido a un bug de Hibernate, si la base de datos utilizada es MySQL, es necesario asignar a este parámetro el valor *3.6.9.Final* (correspondiente a una versión anterior a la aparición del bug), hasta que el bug sea corregido, para evitar problemas serios al manejar ciertos tipos de datos.
Devuelve: nada.

@CliAvailabilityIndicator("roo++ project web deploy") **public boolean isWebDeployAvailable();**
Indica si es posible ejecutar el comando project web deploy en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ project web deploy", help = "Configures the web module to use tomcat and jetty. If a database is to be used you must run \"roo++ project web jpa\" before that") **public void webDeploy();**
Configura el módulo web del proyecto que tiene el foco para el uso de los servidores web Tomcat y Jetty.
Devuelve: nada.

6.2 ProjectOps.java

Operaciones relacionadas con la creación y configuración de proyectos.

void createModules(String projectName, String version, String internalPath);
Crea los diferentes módulos del proyecto Maven/Spring Roo que tiene el foco.
projectName: nombre del proyecto que tiene el foco.
version: número de versión que se asignará a los diferentes módulos y al proyecto completo.
internalPath: ruta interna, dentro del directorio "resources" del JAR, del directorio donde se encuentran realmente todos los resources.
Devuelve: nada.

boolean areModulesCreated();
Determina si los diferentes módulos del proyecto que tiene el foco ya han sido creados, es decir, si ya se ha ejecutado el comando que los crea.
Devuelve: true si los módulos del proyecto ya han sido creados; false en caso contrario.

boolean isLicenseIntroduced();

Indica si ya se ha configurado la licencia en el proyecto que tiene el foco.
Devuelve: true si la licencia ya ha sido configurada; false en caso contrario.

void license(String internalPath);
Configura la licencia en el proyecto que tiene el foco.
internalPath: ruta interna, dentro del directorio "resources" del JAR, del directorio donde se encuentran realmente todos los resources.
Devuelve: nada.

boolean isWebConfigured();
Determina si ya se han ejecutado las acciones de configuración sobre el módulo web del proyecto que tiene el foco.
Devuelve: true si ya se ha ejecutado la configuración; false en caso contrario.

void webConfiguration(String internalPath);
Realiza diferentes acciones de configuración sobre el módulo web del proyecto que tiene el foco.
Devuelve: nada.

boolean isWebJpa();
Determina si el módulo web del proyecto que tiene el foco ya ha sido configurado para el uso de una base de datos.
Devuelve: true si el módulo web ha sido configurado para el uso de una base de datos, false en caso contrario.

void webJpa(String dbms, String databaseName, String user, String password, String internalPath, String server, String port, String hibernateVersion);
Configura el módulo "web" del proyecto que tiene el foco para el uso de una base de datos.
dbms: nombre del sistema de gestión de bases de datos: oracle, mysql, mssql, postgres.
databaseName: nombre de la base de datos (que será el nombre del proyecto si no se indica; no se utiliza en el caso de Oracle).
user: nombre de usuario para el acceso a la base de datos.

password: contraseña para el acceso a la base de datos.
internalPath: ruta interna, dentro del directorio "resources" del JAR, del directorio donde se encuentran realmente todos los resources.
server: máquina que contiene la base de datos.
port: puerto a través del cual se accede a la base de datos.
hibernateVersion: versión de Hibernate a utilizar. Este parámetro no estaba previsto inicialmente; debido a un bug de Hibernate, si la base de datos utilizada es MySQL, es necesario asignar a este parámetro el valor *3.6.9.Final* (correspondiente a una versión anterior a la aparición del bug), hasta que el bug sea corregido, para evitar problemas serios al manejar ciertos tipos de datos.
Devuelve: nada.

boolean isWebDeployed();
Determina si ya se ha configurado el modulo web para el uso de los servidores web Tomcat y Jetty.
Devuelve: true si ya se ha configurado el uso de Tomcat y Jetty; false en caso contrario.

void webDeploy(String internalPath);
Configura el módulo web del proyecto que tiene el foco para el uso de los servidores web Tomcat y Jetty.
internalPath: ruta interna, dentro del directorio "resources" del JAR, del directorio donde se encuentran realmente todos los resources.
Devuelve: nada.

6.3 ProjectOpsImpl.java

Implementación de los métodos declarados en *ProjectOps.java*.

6.4 package-info.java

Declaración del paquete para la creación y configuración de proyectos Spring Roo multimódulo.

7. Usermgmt

Susbsistema que hace posible la gestión de usuarios en la aplicación en desarrollo. Esta gestión de usuarios puede llevarse a cabo de 2 modos: a través de la plataforma software de la organización (en cuyo caso ésta deberá haber sido previamente configurada en la aplicación desarrollada), o bien mediante el uso de las librerías de Spring Security, que permiten la comunicación con un sistema CAS (Central Authentication Service) que se encarga de la autenticación de los usuarios.

Las interfaces de la clase encargada de llevar a cabo la determinación de la disponibilidad y la ejecución de comandos son externas, y están destinadas a ser usadas por Roo, según las anotaciones Java que así se lo indican, mientras que las de las clases de operaciones son internas, destinadas a

ser usadas desde la primera.

7.1 UsermgmtCommands.java

Comandos relacionados con la configuración de la gestión de usuarios en los proyectos.

@CliAvailabilityIndicator("roo++ usermgmt setup") **public boolean isUsermgmtSetupAvailable();**
Indica si es posible ejecutar el comando usermgmt setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

@CliCommand(value = "roo++ usermgmt setup", help = "Configures the project to use an authentication server") **public void usermgmtSetup();**
Configura el módulo web del proyecto que tiene el foco para el uso de un servicio de autenticación de usuarios, sea mediante un módulo externo a Roo (perteneciente a la plataforma utilizada) o mediante el uso de las librerías de Spring Security, según lo que haya establecido el usuario.
Devuelve: nada.

7.2 UsermgmtOperations.java

Operaciones relacionadas con la configuración de la gestión de usuarios en los proyectos.

void usermgmtSetup(String internalPath);
Configura el módulo web del proyecto que tiene el foco para el uso de un servicio de autenticación de usuarios, sea mediante un módulo externo a Roo (perteneciente a la plataforma utilizada) o mediante el uso de las librerías de Spring Security, según lo que haya establecido el usuario.
internalPath: ruta interna, dentro del directorio "resources" del JAR, del directorio donde se encuentran realmente todos los resources.
Devuelve: nada.

boolean canUsermgmtSetup();
Indica si es posible ejecutar el comando usermgmt setup en el proyecto que tiene el foco.
Devuelve: true si es posible ejecutar el comando en este momento; false en caso contrario.

7.3 UsermgmtOperationsImpl.java

Implementación de los métodos declarados en *UsermgmtOperations.java*.

7.4 package-info.java

Declaración del paquete para la configuración de la gestión de usuarios.

8. Utils

Subsistema que proporciona un conjunto de utilidades de todo tipo, destinadas a ser usadas por el resto de susbsistemas, o por él mismo.

Generalmente, las interfaces proporcionadas por las clases CreateModules, UsermgmtUtils y WebModuleUtils son externas (destinadas a ser utilizadas por el resto de subsistemas), mientras que las proporcionadas por las clases FileUtilities, PomUtils y XmlUtils son internas (es decir, destinadas a ser usadas dentro del mismo subsistema Utils), si bien hay excepciones. De todos modos, aunque algunas de las clases son mucho más genéricas que el resto (como FileUtilitites, PomUtils o XmlUtils), lo que hace que sus interfaces, al realizar operaciones de más bajo nivel, tiendan a ser internas, cualquier interfaz de éste susbsistema puede ser tanto interna como externa, es decir, podría ser utilizada tanto desde dentro como desde fuera del susbsistema, aun cuando en la implementación existente no sea así.

8.1 CreateModules.java

Utilidades para la creación de los diferentes módulos de la aplicación.

public static void copyResources(String projectName, String path, String orgPath, ClassLoader cl);
Copia los resources indicados en los ficheros dirs.txt y textfiles.txt, que contienen los directorios, y archivos de texto, respectivamente, que serán copiados de los resources al directorio del proyecto en desarrollo. Cada elemento ha de encontrarse en una linea diferente, e indicar el path completo dentro del directorio del proyecto (sin "/" inicial).
projectName: nombre del proyecto.
path: path del directorio en que se encuentra ubicado el proyecto.
orgPath: path interno en el que se encuentran los resources de este proyecto, dentro del directorio de este nombre (acabado en "/").
cl: ClassLoader correspondiente al hilo en que se produce la ejecución.
Devuelve: nada.

public static void replacePomVariables(String[] poms, String projectName, String version, FileManager fileManager, PathResolver pathResolver, ProjectOperations projectOperations);
Establece, en los ficheros pom.xml indicados, los groupId, artifactId y version, con sus valores correspondientes, incluyendo estos mismos atributos para el parent del módulo, salvo el caso del pom raíz. El foco ha de estar puesto en el módulo web para que este método funcione de la forma prevista.
poms: array con los nombres de los diferentes módulos, mas uno vacío correspondiente a la raíz del proyecto. Las variables seran reemplazadas en cada elemento de este array.

projectName: nombre del proyecto.
version: version del proyecto.
fileManager: objeto FileManager procedente de la clase que invoca a este método.
pathResolver: objeto PathResolver procedente de la clase que invoca a este método.
projectOperations: objeto ProjectOperations procedente de la clase que invoca a este método.
Devuelve: nada.

public static void configureProjectParent(PathResolver pathResolver, ProjectOperations projectOperations);
Establece en el fichero pom.xml de la raíz del proyecto el groupId, artifactId y version del parent del mismo, siempre que exista, según los valores configurados por el usuario. El foco ha de estar puesto en el módulo web para que este método funcione de la forma prevista.
pathResolver: objeto PathResolver procedente de la clase que invoca a este método.
projectOperations: objeto ProjectOperations procedente de la clase que invoca a este método.
Devuelve: nada.

public static void createModule(String moduleName, String projectName, String version, FileManager fileManager, ProjectOperations projectOperations, MavenOperations mavenOperations, PackagingProviderRegistry packagingProviderRegistry);
Crea un módulo con el nombre indicado, invocando a Spring Roo.
moduleName: nombre del módulo a crear.
projectName: nombre del proyecto al que pertenecera dicho módulo.
version: nombre (generalmente un número) de la versión del módulo a crear.
fileManager: objeto FileManager de la clase que invoca al método, necesario para forzar determinadas operaciones de Entrada/Salida, para asegurarse de que todos los cambios realmente han sido guardados en disco.
projectOperations: objeto ProjectOperations de la clase que invoca al método, necesario para conocer qué paquete tiene el foco.
mavenOperations: objeto MavenOperations que sera utilizado para la creación del módulo propiamente dicha.

packagingProviderRegistry: objeto PackagingProviderRegistry, para obtener el tipo de empaquetamiento jar.
Devuelve: nada.

public static void jpaSetupModule(String database, FileManager fileManager, JpaOperations jpaOperations, String moduleName);
Configura un módulo para el uso de una base de datos mediante JPA.
database: nombre del sistema de gestión de bases de datos a utilizar (mssql, mysql, postgres, u oracle; este último también se utilizará si se indica cualquier otro valor.
fileManager: objeto FileManager de la clase que invoca al método, necesario para forzar determinadas operaciones de Entrada/Salida, para asegurarse de que todos los cambios realmente han sido guardados en disco.
jpaOperations: objeto JpaOperations de la clase que invoca al método, necesario para configurar el acceso del módulo a la base de datos.
moduleName: nombre del módulo que se conectará a la base de datos mediante JPA.
Devuelve: nada.

public static void webSetupModule(String moduleName, String projectName, FileManager fileManager, JspOperations ops);
Configura un módulo para que ofrezca una interfaz web.
moduleName: nombre del módulo que tendrá una interfaz web.
projectName: nombre del proyecto al que pertenece dicho módulo.
fileManager: objeto FileManager de la clase que invoca al método, necesario para forzar determinadas operaciones de Entrada/Salida, para asegurarse de que todos los cambios realmente han sido guardados en disco.
ops: objeto JspOperations de la clase que invoca al método, que será utilizado para crear la interfaz web del módulo.
Devuelve: nada.

8.2 EarPackaging.java

Declaración de un packaging de nombre ear, no soportado nativamente por Roo, para posibilitar su uso en el módulo “Ear” del proyecto desarrollado.

8.3 FileUtilities.java

Utilidades para la gestión de ficheros.

public static String readFileAsStringThrows(String filePath) throws IOException;
Lee un fichero de texto, devolviendo su contenido en un String, y lanzando una excepción si el fichero indicado no existe.
filePath: ruta completa del fichero a leer.
Devuelve: el contenido del fichero indicado.

public static String readFileAsString(String filePath);
Lee un fichero de texto, devolviendo su contenido en un String, o la cadena vacía si el fichero indicado no existe.
filePath: ruta completa del fichero a leer.
Devuelve: el contenido del fichero indicado, o la cadena vacía si éste no existe.

public static void replaceText(String oldText, String newText, String filePath);
Reemplaza en un fichero de texto todas las ocurrencias de una cadena por otra.
oldText: cadena a reemplazar.
newText: cadena que reemplazará a la anterior.
filePath: ruta completa del fichero de texto en el que se reemplazarán las cadenas.
Devuelve: nada.

public static String getConfigurationFilePath(String internalFilePath);
Devuelve la ruta completa de un archivo o directorio de configuración. Todos ellos se encuentran en el directorio "RooAddonsData", que puede estar situado en el lugar indicado por la variable de entorno "ROOADDON" del sistema operativo, o, si ésta no está definida, contenida en el directorio "home" del usuario.
internalFilePath: ruta del archivo o directorio de configuración, tomando como raíz el directorio "RooAddonsData".
Devuelve: ruta completa del archivo o directorio de configuración.

public static void removeBlankLines(String path);
Elimina todas las líneas vacías de un fichero de texto, es decir, todas las líneas que no tengan ningún carácter o que sólo tengan tabuladores y/o espacios en blanco.

path: ruta completa del fichero de texto del que se eliminarán las líneas vacías.
Devuelve: nada.

public static void copyDirContent(String dirOrigen, String dirDestino);
Copia todo el contenido de un directorio a otro directorio.
dirOrigen: ruta completa del directorio a copiar.
dirDestino: ruta completa del directorio al que se copiará el contenido del anterior.
Devuelve: nada.

public static String readTextFile(String internalFilePath, String internalPath, ClassLoader cl);
Obtiene el contenido de un fichero de texto situado en el directorio "resources" del JAR.
internalFilePath: ruta del fichero de texto dentro de la raíz de "resources" (indicada en el siguiente parámetro).
internalPath: ruta de la raiz de "resources", donde se encuentran realmente todos los recursos allí almacenados.
cl: objeto ClassLoader necesario, para acceder a los "resources" del JAR.
Devuelve: el contenido del archivo de texto, perteneciente a los "resources" del JAR, indicado.

public static String getFirstLineConfFile(String confFilePath) throws IOException;
Obtiene la primera línea de un archivo de configuración, lanzando una excepción si dicho archivo no existe o no tiene contenido.
confFilePath: ruta del archivo, dentro del directorio de configuración.
Devuelve: la primera línea del archivo de configuración indicado.

public static void problemClosingFile();
Informa al usuario de que se ha producido un error al tratar de cerrar un fichero.
Devuelve: nada.

public static void problemCreatingDir(String dir);
Informa al usuario de que se ha producido un error al tratar de crear un directorio.

dir: ruta y nombre del directorio cuya creación ha fallado.
Devuelve: nada.

8.4 PomUtils.java

Utilidades para la edición de ficheros pom.xml de Maven.

public static void replaceVariables(String pomPath, String moduleName, String projectName, String version, String topLevelPackage);
Asigna los valores correspondientes a las diferentes variables en un fichero pom.xml: nombre, groupId, artifactId y version del proyecto y de su parent.
pomPath: ruta completa del fichero pom.xml en el que se asignarán los valores a las variables.
moduleName: nombre del módulo al que pertenece dicho pom.xml.
projectName: nombre del proyecto al que pertenece dicho pom.xml.
version: versión del proyecto, y por tanto de sus módulos.
topLevelPackage: paquete Java correspondiente al módulo (es decir, groupId).
Devuelve: nada.

public static void addDependency(String pomPath, String groupId, String artifactId, String version) throws XpathExpressionException, ParserConfigurationException, SAXException, IOException, TransformerException;
Introduce como hija del nodo "dependencies", hijo de la raiz del pom.xml ("project"), una nueva dependencia, lanzando excepciones si el fichero no existe o no tiene los nodos “project” y/o “dependencies”.
pomPath: ruta completa del fichero pom.xml en el que se creará la nueva dependencia.
groupId: groupId de la nueva dependencia.
artifactId: artifactId de la nueva dependencia.
version: versión de la nueva dependencia.
Devuelve: nada.

public static void addDependencyToBuildPlugin(String pomPath, String pluginArtifactId,

String groupId, **String artifactId,** **String version)** **throws XpathExpressionException,** **ParserConfigurationException,** **SAXException,** **IOException,** **TransformerException;**
Introduce una nueva dependencia en uno de los "build/plugins" de un fichero pom.xml, lanzando una excepción si el fichero pom.xml indicado no existe, no tiene el contenido adecuado, o si no existe el plugin indicado dentro del mismo.
pomPath: ruta completa del fichero pom.xml en el que se introducirá la dependencia del plugin.
pluginArtifactId: artifactId del plugin al que se pondrá la dependencia.
groupId: groupId de la dependencia a introducir.
artifactId: artifactId de la dependencia a introducir.
version: versión de la dependencia a introducir.
Devuelve: nada.

public static void addScope(String value, String artifactIdxPath, **Document doc, String pomPath)** **throws XpathExpressionException,** **TransformerException;**
Introduce un elemento “<scope>” como hijo del nodo indicado por el XPath de su artifactId hijo.
value: valor (contenido) del nuevo nodo XML “<scope>”.
artifactIdxPath: XPath del artifactId del elemento al que se introducirá el nodo “<scope>” como hijo.
doc: documento XML en el que se introducirá el nuevo nodo.
pomPath: path del fichero pom.xml en el que se introducirá el nuevo nodo.
Devuelve: nada.

public static void addProperty(String pomPath, String name, String value) **throws XpathExpressionException,** **ParserConfigurationException,** **SAXException,** **IOException;**
Introduce una nueva propiedad en el campo "properties" del fichero pom.xml indicado.

pomPath: ruta completa del fichero pom.xml en el que se introducirá la propiedad.
name: nombre de la propiedad a introducir.
value: valor de la propiedad a introducir.
Devuelve: nada.

public static void setVersion(String pomPath, String version, FileManager fileManager);
Establece en el pom.xml (indicado por el parámetro pomPath) la versión del proyecto, y por tanto, del módulo.
pomPath: ruta completa del fichero pom.xml en el que se establecerá la versión.
version: nombre (normalmente, con número/s) de la versión a establecer.
fileManager: objeto FileManager necesario para asegurar que las operaciones de entrada/salida realmente han sido escritas en disco.
Devuelve: nada.

public static void addLicensePlugin(String pomPath);
Introduce en el pom.xml indicado el plugin de la licencia contenido en el fichero plugin-license.xml del directorio de configuración.
pomPath: ruta completa del pom.xml en el que se introducira el plugin de la licencia
Devuelve: nada.

public static void addLicense(String pomPath);
Introduce en el pom.xml indicado la licencia según el contenido del fichero de configuración licenses.xml.
pomPath: ruta completa del pom.xml en el que se introducirá la licencia.
Devuelve: nada.

public static void addBuildPlugin(String pomPath, Node pluginAImportar);
Introduce en el fichero pom.xml indicado, en project/build/plugins, el nodo (conteniendo un plugin) indicado como parámetro Si no existieran los elementos build y/o plugins en el pom.xml, son creados.
pomPath: ruta completa del pom.xml en el que se introducirá el plugin.

pluginAImportar: nodo XML que contiene el plugin a importar.
Devuelve: nada.

public static void addPlatformDependency(String module, **ProjectOperations projectOperations,** **FileManager fileManager,** **String platformName,** **String groupId, String testModule)** **throws XpathExpressionException,** **ParserConfigurationException,** **SAXException,** **IOException,** **TransformerException ;**
Introduce en un pom la dependencia con respecto al módulo del mismo nombre de la plataforma a utilizar, y al módulo de test de dicha plataforma que siempre es utilizado, lanzando una excepción si l pom.xml del módulo indicado no existe en el lugar esperado o no tiene el contenido adecuado.
module: nombre del módulo al que se le pondra la dependencia.
projectOperations: objeto ProjectOperations de la clase que invoca al método, que permite conocer qué módulo y proyecto tienen el foco.
fileManager: objeto FileManager necesario para poder garantizar que los cambios sobre los ficheros realmente se han escrito en disco.
platformName: nombre de la plataforma.
groupId: groupId de la plataforma.
testModule: nombre del módulo de test de la plataforma.
Devuelve: nada.

8.5 ProjectUtils.java

Utilidades para la gestión de proyectos Maven/Spring Roo.

public static void copyBinaryFile(String projectPath, String projectName, **String internalFilePath,** **String internalPath, ClassLoader cl);**
Copia un fichero binario desde el directorio "resources" del JAR hasta el proyecto.
projectPath: ruta completa del directorio raíz del proyecto.
projectName: nombre del proyecto.
internalFilePath: ruta interna (incluyendo el nombre) del fichero a copiar, que sera la misma tanto en el "resources" del JAR como dentro del proyecto (es decir, un fichero debe encontrarse en "resources" en la misma ubicación a la que será copiado dentro del proyecto, excepto por el

nombre del directorio del módulo que en el proyecto incluirá el nombre de este último).
internalPath: ruta de la raíz del directorio de resources del JAR.
cl: objeto ClassLoader necesario para obtener el fichero contenido en "resources" en el JAR.
Devuelve: nada.

public static void copyExternBinaryFile(String projectPath, String projectName, String internalFilePath, String filePath);
Copia un fichero binario al proyecto.
projectPath: ruta completa del directorio raíz del proyecto.
projectName: nombre del proyecto.
internalFilePath: ruta destino del fichero a copiar, partiendo del directorio del módulo que lo contiene (por ejemplo, "web/fichero.pdf", sin incluir en ella el nombre del proyecto).
filePath: ruta completa del fichero a copiar.
Devuelve: nada.

public static void copyTextFile(String projectPath, String projectName, String internalFilePath, String internalPath, ClassLoader cl);
Copia un fichero de texto desde el directorio "resources" del JAR hasta el proyecto.
projectPath: ruta completa del directorio raíz del proyecto.
projectName: nombre del proyecto.
internalFilePath: ruta interna (incluyendo el nombre) del fichero a copiar, que sera la misma tanto en el "resources" del JAR como dentro del proyecto (es decir, un fichero debe encontrarse en "resources" en la misma ubicación a la que será copiado dentro del proyecto, excepto por el nombre del directorio del módulo que en el proyecto incluirá el nombre de este último).
internalPath: ruta de la raíz del directorio de resources del JAR.
cl: objeto ClassLoader necesario para obtener el fichero contenido en "resources" en el JAR.
Devuelve: nada.

public static void copyExternTextFile(String projectPath, String projectName, String internalFilePath,

String filePath);
Copia un fichero de texto al proyecto.
projectPath: ruta completa del directorio raíz del proyecto.
projectName: nombre del proyecto.
internalFilePath: ruta destino del fichero a copiar, partiendo del directorio del módulo que lo contiene (por ejemplo, "web/fichero.txt", sin incluir en ella el nombre del proyecto).
filePath: ruta completa del fichero a copiar.
Devuelve: nada.

public static void appendTextToFile(String text, String projectPath, String projectName, String internalFilePath);
Introduce un texto en la parte final de un fichero de texto, dentro de un proyecto.
text: texto a introducir.
projectPath: ruta del directorio raíz del proyecto.
projectName: nombre del proyecto.
internalFilePath: ruta del fichero en el que se introducira el texto, partiendo del directorio del módulo que lo contiene (por ejemplo, "web/fichero.txt", sin incluir en ella el nombre del proyecto).
Devuelve: nada.

public static String getProjectPath(ProjectOperations projectOperations);
Obtiene la ruta completa del directorio raíz del proyecto. Sólo funcionará de la forma esperada si el foco se encuentra en la raíz de dicho proyecto, y no en uno de sus módulos.
projectOperations: objeto ProjectOperations de la clase que invoca al método, necesario para poder determinar la ubicación del proyecto.
Devuelve: ruta completa del directorio raíz del proyecto.

public static boolean areModulesCreated(ProjectOperations projectOperations);
Determina si los diferentes módulos han sido creados en el proyecto (es decir, si se ha ejecutado create-modules). No funcionará de la forma esperada si se han introducido manualmente cambios inapropiados en el pom.xml raíz del proyecto. Tampoco detectará si se han eliminado a mano uno o varios de los módulos previamente creados.

projectOperations: objeto ProjectOperations de la clase que invoca al método, necesario para poder determinar la ubicación del proyecto.
Devuelve: true en caso de ya que se hayan creado los módulos del proyecto; false en caso contrario.

public static void setMavenPlugins(ProjectOperations projectOperations);
Configura los plugins maven de tomcat y jetty en el fichero pom.xml del módulo web, de forma que utilicen la base de datos indicada en el mismo fichero, caso de que haya alguna.
projectOperations: objeto ProjectOperations de la clase que invoca al método, necesario para poder determinar la ubicación del proyecto.
Devuelve: nada.

public static void undoWebDeploy(ProjectOperations projectOperations);
Elimina las dependencias de los plugins maven de tomcat y jetty en el fichero pom.xml del módulo web del proyecto, de forma que pueda volver a realizarse un "web deploy" sin que haya elementos repetidos u otros problemas relacionados.
projectOperations: objeto ProjectOperations de la clase que invoca al método, necesario para poder determinar la ubicación del proyecto.
Devuelve: nada.

public static void dataSourceWebXml(String projectPath, String internalPath, ProjectOperations projectOperations, ClassLoader cl);
Introduce en el fichero "src/main/webapp/WEB-INF/web.xml", dentro del módulo "web" del proyecto, la referencia al DataSource correspondiente, según el nombre del proyecto, para que Jetty funcione correctamente.
projectPath: ruta del directorio raiz del proyecto.
internalPath: ruta de la raiz de los resources dentro del JAR.
projectOperations: objeto ProjectOperations necesario para conocer la ubicacion de los archivos del proyecto.
cl: objeto ClassLoader necesario para obtener los archivos contenidos en el directorio "resources" del JAR.
Devuelve: nada.

8.6 UsermgmtUtils.java

Utilidades para ayudar a introducir la gestión de usuarios en un proyecto.

public static void usermgmtSetupCustom(String internalPath, String projectPath, ProjectOperations projectOperations, ClassLoader cl);
Configura la gestión de usuarios en el proyecto, haciendo uso de un servicio remoto, mediante un módulo de gestión de usuarios contenido en el mismo.
internalPath: ruta de la raíz del directorio de resources del JAR.
projectPath: ruta del directorio raíz del proyecto.
projectOperations: objeto ProjectOperations de la clase que invoca al método, necesario para poder determinar la ubicación del proyecto.
cl: objeto ClassLoader necesario para acceder a los elementos contenidos en "resources" en el JAR.
Devuelve: nada.

public static void usermgmtSetupSpring(String internalPath, String projectPath, ProjectOperations projectOperations, ClassLoader cl);
Configura la gestión de usuarios en el proyecto, haciendo uso de las librerías de seguridad que proporciona Spring para este fin (Spring Security).
internalPath: ruta de la raíz del directorio de resources del JAR.
projectPath: ruta del directorio raíz del proyecto.
projectOperations: objeto ProjectOperations de la clase que invoca al método, necesario para poder determinar la ubicación del proyecto.
cl: objeto ClassLoader necesario para acceder a los elementos contenidos en "resources" en el JAR.
Devuelve: nada.

8.7 WebModuleUtils.java

Utilidades para la realización de diversas operaciones sobre el modulo "web" de una aplicación multimódulo desarrollada con Spring Roo.

public static void copyApplicationPropertiesToWebModule(String projectPath, String internalPath, String projectName,

ClassLoader cl);
Copia el fichero application.properties desde el directorio de "resources" del JAR hasta el módulo "web" del proyecto en desarrollo.
projectPath: path del directorio raíz del proyecto en desarrollo.
internalPath: path interno del directorio de resources del JAR.
projectName: nombre del proyecto en desarrollo.
cl: ClassLoader de la clase que invoca a este método, necesario para obtener los resources del JAR.
Devuelve: nada.

public static void copyDatabasePropertiesToWebModule(String projectPath, String internalPath, String projectName, String databaseName, ClassLoader cl);
Copia el fichero database.properties desde el directorio de "resources" del JAR hasta el módulo "web" del proyecto en desarrollo.
projectPath: path del directorio raíz del proyecto en desarrollo.
internalPath: path interno del directorio de resources del JAR.
projectName: nombre del proyecto en desarrollo.
databaseName: nombre de la base de datos a utilizar por el proyecto en desarrollo.
cl: ClassLoader de la clase que invoca a este método, necesario para obtener los resources del JAR.
Devuelve: nada.

public static void copyApplicationContextPlatformToWebModule(String projectPath, String projectName);
Copia el fichero "applicationContext-platform.xml" desde el directorio de configuración del usuario hasta el proyecto indicado.
projectPath: path del directorio raíz del proyecto.
projectName: nombre del proyecto.
Devuelve: nada.

public static void copyApplicationContextSecurityToWebModule(String projectPath, String projectName);
Copia el fichero "applicationContext-security.xml" desde el directorio de configuración del usuario hasta el proyecto indicado.
projectPath: path del directorio raíz del proyecto.
projectName: nombre del proyecto.
Devuelve: nada.

public static void copyApplicationContextPortalToWebModule(String projectPath, String projectName);
Copia el fichero "applicationContext-portal.xml" desde el directorio de configuración del usuario hasta el proyecto indicado.
projectPath: path del directorio raíz del proyecto.
projectName: nombre del proyecto.
Devuelve: nada.

public static void replaceHeader(String projectPath, String projectName);
Reemplazar el fichero header.jspx del módulo web del proyecto indicado por uno adaptado al uso del portal utilizado, que el usuario ha introducido en su directorio de configuración.
projectPath: path del directorio raíz del proyecto.
projectName: nombre del proyecto.
Devuelve: nada.

public static void updateAppContextWebMvcConfig(String projectPath, String internalPath, String projectName, ClassLoader cl);
Introduce los beans messageSource, applicationPropertiesBean y property-placeholder (ambos obtenidos de los "resources" del JAR) en el fichero applicationContext.xml, y el último de ellos también en webmvc-config.xml dentro del módulo web de un proyecto.
projectPath: path del directorio raíz del proyecto.
internalPath: path interno del directorio de resources del JAR.
projectName: nombre del proyecto.

cl: objeto ClassLoader necesario para extraer el contenido de "resources" en el JAR.
Devuelve: nada.

public static void updateAppContextDatabase(String projectPath, String internalPath, String projectName, ClassLoader cl);
Introduce en el fichero applicationContext.xml del módulo web de un proyecto el bean entityManagerFactory y el dataSource de jndi, necesarios para el uso de una base de datos en el proyecto.
projectPath: path del directorio raíz del proyecto.
internalPath: path interno del directorio de resources del JAR.
projectName: nombre del proyecto.
cl: objeto ClassLoader necesario para extraer el contenido de "resources" en el JAR.
Devuelve: nada.

public static void setMavenPlugins(String driver, String pomPath);
Configura los plugins de Maven para Tomcat y Jetty en el fichero pom.xml del módulo web de un proyecto, introduciendo sus dependencias y número de versión correspondientes, y, si el nombre del plugin para Jetty era "jetty-maven-plugin", cambiándolo a "maven-jetty-plugin", que es lo que corresponde para la versión utilizada (6.1.18).
driver: nombre del driver de bases de datos utilizado (ojdbc, mysql, postgresql, jtds).
pomPath: ruta completa del fichero pom.xml en el que se configurarán los plugins.
Devuelve: nada.

public static void renamePersistence(String projectPath, String projectName);
Cambia el nombre del fichero persistence.xml a persistenceInfo.xml, dentro del módulo web de un proyecto, y establece en el mismo el valor de "persistence-unit" de acuerdo al nombre del proyecto.
projectPath: ruta completa del directorio raíz del proyecto.
projectName: nombre del proyecto.
Devuelve: nada.

public static void configureEclipsePlugin(String projectPath,

String projectName);
Configura el plugin Maven de Eclipse del pom.xml del módulo web de un proyecto para que el campo "useProjectReferences" tenga el valor "false".
projectPath: ruta completa del directorio raíz del proyecto.
projectName: nombre del proyecto.
Devuelve: nada.

public static void addLicenseToWebModule(String projectPath, String internalPath, String projectName, ClassLoader cl);
Introduce en el fichero pom.xml del módulo web de un proyecto, el plugin de Maven correspondiente a las licencias utilizadas.
projectPath: path del directorio raíz del proyecto.
internalPath: path interno del directorio de resources del JAR.
projectName: nombre del proyecto.
cl: objeto ClassLoader necesario para extraer el contenido de "resources" en el JAR.
Devuelve: nada.

public static void addSourcePlugin(String webPomPath, String internalPath, ClassLoader cl);
Introduce el contenido del fichero "plugin-source.xml" de los resources del JAR, como plugin en el pom.xml del módulo web de un proyecto.
webPomPath: ruta completa del fichero pom.xml del módulo web de un proyecto.
internalPath: path interno del directorio de resources del JAR.
cl: objeto ClassLoader necesario para extraer el contenido de "resources" en el JAR.
Devuelve: nada.

public static void copyJettyTomcatConf(String projectPath, String internalPath, String dbms, String database, String user, String password, String server, String port, String projectName, ClassLoader cl);
Copia al módulo web de un proyecto los directorios y ficheros de configuración necesarios para el

despliegue del mismo con los servidores web Jetty y Tomcat.
projectPath: ruta completa del directorio raíz del proyecto.
internalPath: path interno del directorio de resources del JAR.
dbms: nombre del sistema de gestion de bases de datos utilizado por la base de datos (mysql, mssql, postgres u oracle).
database: nombre de la base de datos utilizada (no confundir con sistema de gestión de bases de datos).
user: nombre de usuario con el que acceder a la base de datos.
password: contraseña correspondiente a dicho usuario, para el acceso a la base de datos.
server: dirección IP o nombre (incluyendo dominio, si es necesario) de la máquina en la que se encuentra la base de datos.
port: puerto en el que se está ejecutando el sistema de gestion de bases de datos en la máquina correspondiente (si se indica la cadena vacía, se asignará el número de puerto de la instalación por defecto del sistema de gestión de bases de datos utilizado).
projectName: nombre del proyecto.
cl: objeto ClassLoader necesario para extraer resources del JAR.
Devuelve: nada.

public static void copyJettyConfWithoutDatabase(String projectPath, String projectName, String internalPath, ClassLoader cl);
Crea la estructura de directorios y copia los ficheros relativos a la configuración de Jetty desde los resources del JAR hasta el módulo web de un proyecto, eliminando la parte relativa a la configuración de la base de datos en "jetty.xml".
projectPath: ruta del directorio raíz del proyecto.
projectName: nombre del proyecto.
internalPath: path interno del directorio de resources del JAR.
cl: objeto ClassLoader necesario para extraer resources del JAR.
Devuelve: nada.

public static void setJavaVersion(String version, ProjectOperations projectOperations);

Establece en el archivo pom.xml del módulo web de un proyecto la versión de Java a utilizar, en formato 1.x (por ejemplo, 1.6 en lugar de 6).
version: versión de Java a utilizar (por ejemplo, 1.6, pero no 6).
projectOperations: objeto projectOperations de la clase que invoca al método, necesario para saber qué proyecto tiene el foco.
Devuelve: nada.

public static boolean isWebJpa(ProjectOperations projectOperations);
Determina si el módulo web del proyecto que tiene el foco ya ha sido configurado para el uso de una base de datos.
projectOperations: objeto projectOperations de la clase que invoca al método, necesario para saber qué proyecto tiene el foco.
Devuelve: true si el módulo web ha sido configurado para el uso de una base de datos, false en caso contrario.

public static void setHibernateVersion(String version, ProjectOperations projectOperations);
Establece la versión de Hibernate a utilizar, en el archivo pom.xml del módulo web del proyecto que tiene el foco.
version: versión de Hibernate a utilizar.
projectOperations: objeto projectOperations de la clase que invoca al método, necesario para saber qué proyecto tiene el foco.
Devuelve: nada.

public static File getWebModuleFile(String file, ProjectOperations projectOperations);
Obtiene una referencia a un fichero existente dentro del módulo web del proyecto que tiene el foco.
file: ruta interna del fichero, tomando como raíz la del módulo web.
projectOperations: objeto projectOperations de la clase que invoca al método, necesario para saber qué proyecto tiene el foco.
Devuelve: referencia al fichero indicado, o null si éste o el módulo web no existen.

8.8 XmlUtils.java

Utilidades para el manejo de ficheros XML.

public static void addChildToRoot(String xmlPath, String nodeName, String nodeValue);
Introduce un nuevo nodo como hijo de la raíz de un documento XML.
xmlPath: ruta completa del fichero XML en que se introducirá el nuevo nodo.
nodeName: nombre del nuevo nodo a introducir.
nodeValue: contenido del nuevo nodo a introducir.
Devuelve: nada.

public static void addChildToNode(String xmlPath, String parentXPath, String nodeName, String nodeValue);
Introduce un nuevo nodo como hijo de otro nodo existente en un documento XML.
xmlPath: ruta completa del fichero XML en que se introducirá el nuevo nodo.
parentXPath: XPath del nodo del que "colgara" el nuevo nodo.
nodeName: nombre del nuevo nodo a introducir.
nodeValue: contenido del nuevo nodo a introducir.
Devuelve: nada.

public static void changeNodeValue(String xmlPath, String nodeXPath, String newValue);
Establece un nuevo valor para el contenido de un nodo en un documento XML.
xmlPath: ruta completa del fichero XML.
nodeXPath: XPath del nodo a modificar.
newValue: nuevo valor para el contenido del nodo.
Devuelve: nada.

public static void setNodeAttribute(String xmlPath, String nodeXPath, String attribute, String value);
Introduce un atributo para un nodo en un documento XML.
xmlPath: ruta completa del fichero XML.
nodeXPath: XPath del nodo en el que se introducirá el atributo.
attribute: nombre del atributo.

value: valor del atributo.
Devuelve: nada.

public static void appendFileToRoot(String xmlPath, String internalPath, String xmlFileToAppend, ClassLoader cl);
Introduce el contenido de un fichero XML perteneciente a los "resources" del JAR como hijo de la raíz de otro documento del mismo tipo.
xmlPath: ruta completa del fichero XML en el que se introducirá el contenido del otro documento.
internalPath: ruta de la raíz del directorio de resources del JAR.
xmlFileToAppend: ruta del documento XML a introducir dentro del otro, partiendo de la raíz de los "resources" del JAR.
cl: objeto ClassLoader necesario para obtener el fichero contenido en "resources" en el JAR.
Devuelve: nada.

public static void eliminaNodoSiExiste(String xmlPath, String nodeXPath);
Elimina un nodo (en caso de que exista) de un documento XML.
xmlPath: ruta completa del fichero XML
nodeXPath: XPath del nodo a eliminar del documento.
Devuelve: nada.

public static void guardaXml(Document doc, String xmlPath) throws TransformerException;
Guarda el contenido de un documento XML en un fichero, lanzando una excepción si no se puede escribir en el fichero indicado, o si el documento tiene un contenido inconsistente.
doc: documento XML a guardar.
xmlPath: fichero en el que se guardará el documento.
Devuelve: nada.

public static NodeList evaluateXPath(String xPath, Document doc) throws XPathExpressionException;
Evalúa una expresión XPath sobre un documento XML, lanzando una excepción si el documento

tiene un contenido inconsistente, o si no es posible evaluar la exprexión XPath.
xPath: expresión XPath a evaluar.
doc: documento XML sobre el que se evaluará la expresión.
Devuelve: Lista de nodos que se ajustan a lo indicado en la expresión XPath.

public static Document createDocumentFromFile(String xmlPath) **throws ParserConfigurationException,** **SAXException,** **IOException;**
Crea un documento XML a partir de un archivo, lanzando una excepción si el archivo indicado no existe o no contiene un documento XML válido.
xmlPath: path del archivo XML a partir del cual se creará el documento.
Devuelve: documento XML contenido en el archivo indicado.

boolean canUsermgmtSetup();
Crea un documento XML a partir de una cadena, lanzando una excepción si la cadena no contiene un documento XML válido.
str: cadena con código XML a partir del que se creará el documento.
Devuelve: documento XML construido a partir del texto de la cadena indicada.

8.9 package-info.java

Declaración del paquete que contiene diferentes clases de utilidades.

Diseño de datos

1. Estructura de un proyecto

Todo proyecto desarrollado con Roo++ es un proyecto Maven multimódulo, compuesto siempre por los mismos 6 módulos (salvo que el usuario posteriormente pueda eliminar alguno de ellos o introducir otros adicionales, pero esto sale del ámbito de Roo++).

Cada uno de estos módulos se encuentra en un directorio, situado dentro del directorio raíz del proyecto. El nombre del directorio que contiene un módulo siempre sigue el patrón *nombreDelProyecto-nombreDelMódulo*, mientras que el directorio raíz del proyecto puede tener cualquier nombre, con el fin de no imponer restricciones en un directorio que no ha sido creado por Roo++ ni tiene por qué influir de ninguna manera en su comportamiento.

A su vez, el directorio de cada módulo contiene 2 subdirectorios: *src* y *target*. El segundo es rellenado automáticamente por Maven durante la compilación del proyecto (o del módulo), y su contenido será de interés únicamente para "recoger" los resultados de ésta, fundamentalmente el archivo EAR que permite el despliegue definitivo de la aplicación, o sus manuales de instalación y de usuario. El contenido del directorio *src* será el que veremos en detalle, para cada uno de los módulos, ya que es el único que es modificado de forma directa por Roo++.

En algunos de los módulos, el directorio *src* contiene un subdirectorio de nombre *main*, en el cual se encuentran todos los archivos de *src* de que trataremos; también puede existir un directorio *tests*, para permitir la realización de pruebas sobre el proyecto, pero esto sale fuera del ámbito de Roo++ y no lo abordaremos.

Finalmente, tanto el directorio raíz del proyecto como los de cada uno de los módulos contienen un archivo *pom.xml,* en el que se definen una serie de parámetros para su compilación por parte de Maven.

Analizaremos el contenido de cada uno de los módulos de un proyecto ya desarrollado con Roo++, y para el que se han ejecutado todos los comandos que éste proporciona.

1.1 Módulo core

La finalidad de este módulo es implementar todas las operaciones de lógica de negocio, o utilidades de cualquier tipo, en la aplicación web desarrollada con Roo++. Por ello, éste tan solo crea el módulo, pero no intruduce ningún archivo en particular en él (tan solo el pom.xml, al igual que en cualquier otro de los módulos del proyecto).

Dentro del directorio raíz de este módulo, los directorios ***src/main/java*** y ***src/main/resources***, están destinados a contener, respectivamente, los archivos .java en que se ha de implementar la funcionalidad proporcionada por este módulo, y los archivos (sean de texto, imágenes, o de cualquier otro tipo) necesarios para ello.

1.2 Módulo docs

Su finalidad es la de generar la documentación del proyecto desarrollado, en formato PDF, a partir de una serie de archivos fuente (XML, PDF, imágenes, fuentes), mediante DocBook. Esta documentación consiste de 2 manuales, uno de instalación y otro de usuario.

Dentro del directorio *src* de este módulo se encuentran los siguientes subdirectorios:

- **assembly:** contiene ciertos parámetros de configuración sobre como se ha de construir éste

módulo.

- **docbkx:** contiene el archivo custom.xsl, que es introducido por Roo++ según haya sido colocado en el directorio de configuración correspondiente, que determina aspectos tales como las fuentes o la distribución de los elementos en la documentación a generar.
- **fonts:** en él introduce Roo++ los archivos que contienen las fuentes (tipografías) que se han de usar en la documentación.
- **front_pages:** contiene 2 archivos PDF, uno por cada manual a generar, con una serie de páginas fijas que serán copiadas "tal cual" a la documentación resultante. El uso exacto de estas páginas dependerá de la configuración establecida.
- **images:** directorio en que se han de introducir, manualmente, las imágenes que sea necesario utilizar en la documentación. Para utilizarlas se han de referenciar desde el archivo XML correspondiente al manual que hace uso de ellas.

Además, en el mismo directorio src existen los siguientes archivos:

- **InstallationManual.xml:** archivo XML con el contenido, en DocBook, del manual de instalación de la aplicación.
- **UserManual.xml:** archivo XML con el contenido, en DocBook, del manual de usuario de la aplicación.

1.3 Módulo ear

La finalidad de este módulo es la de contener, tras su compilación, la aplicación web, dentro del subdirectorio *target*, contenida en un archivo EAR, listo para su despliegue en un servidor web o de aplicaciones (Tomcat, Jetty, JBoss, etc).

Su subdirectorio *src* tan sólo contiene un archivo: **src/main/application/META-INF/weblogic-application.xml**, que configura las prioridades de los paquetes Java de que habitualmente hará uso una aplicación desarrollada con Roo++.

1.4 Módulo package

Su finalidad es únicamente la de empaquetar todo el proyecto desarrollado en un único archivo. Para ello, contiene el archivo **src/main/assembly/package.xml**, en que se definen las reglas para este empaquetamiento.

1.5 Módulo scripts

Su objetivo es el de contener todos los scripts SQL que se han de ejecutar en el sistema de gestión de bases de datos a utilizar, antes de desplegar la aplicación, para que ésta pueda acceder a todas las tablas y campos que necesita. De esta forma los scripts pueden ser distribuidos junto con la aplicación.

Los archivos .sql han de ser colocados (manualmente, salvo el caso de context-dataload.sql, que lo es de forma automática por Roo++ al ejecutar su comando `context setup`) en el directorio ***src/main/resources***.

Por otro lado, también está presente el archivo, introducido por Roo++, **src/assembly/compress-zip.xml**, que define las reglas para el empaquetamiento de estos scripts durante la compilación del módulo.

1.6 Módulo web

Este módulo contiene la aplicación web propiamente dicha, incluyendo desde sus elementos visuales (tales como imágenes u hojas de estilos) hasta la configuración de su despliegue en los servidores web Tomcat y Jetty.

Además del archivo pom.xml, que tienen todos los módulos, contiene los siguientes elementos:

- **src/main/java:** en subdirectorios aquí contenidos se encontrarán los archivos .java y .aj relacionados con las diferentes entidades que maneja la aplicación desarrollada. Estos archivos son creados automáticamente por Roo cuando se crean las entidades o se definen sus campos o relaciones; por tanto, no entran dentro del ámbito de Roo++ en sí mismo.
- **src/main/jetty:** directorio de configuración del despliegue de la aplicación en el servidor web Jetty. Contiene, porque así lo requiere, 3 subdirectorios vacíos (***logs***, ***webapps*** y ***webapps-plus***), un subdirectorio ***etc*** (con 2 archivos de configuración, **realm.properties**, con la definición de usuarios y contraseñas para el uso de Jetty , y **webdefault.xml** , con la configuración de Jetty común a todas las aplicaciones), y el archivo **jetty.xml**, con la configuración general de Jetty, incluyendo la del acceso a la base de datos.
- **src/main/resources/META-INF/persistenceInfo.xml:** contiene la configuración del acceso a los datos, almacenados en la base de datos utilizada por la aplicación, mediante Hibernate, incluyendo la definición del dialecto SQL a utilizar para ello.
- **src/main/resources/META-INF/portal.properties:** contiene propiedades que es necesario establecer para la integración de la aplicación web desarrollada en el portal web de la organización. La ubicación de este archivo en un directorio superior al del resto de archivos .properties se debe a un conflicto con un archivo similar en una de las librerías utilizadas, que impedía su ubicación en el directorio spring, junto con el resto de archivos de este tipo.
- **src/main/resources/META-INF/spring/application.properties:** configura ciertas propiedades generales de la aplicación, como su URL, nombre, tema por defecto, ubicación de los archivos de localización, y nivel de privilegios de un usuario requerido para el acceso a la misma. Todas estas propiedades son introducidas por los comandos de Roo++ que configuran cada uno de estos aspectos del proyecto.
- **src/main/resources/META-INF/spring/database.properties:** contiene la definición de todas las propiedades que es necesario conocer para el acceso a la base de datos por parte de la aplicación, incluyendo el driver a utilizar, datasource, nombre y ubicación de la base de datos, dialecto SQL, y la configuración acerca de si las tablas y campos de la base de datos han de ser creados o no durante el despliegue de la aplicación en el servidor web.
- **src/main/resources/META-INF/spring/applicationContext.xml:** este archivo contiene referencias a los beans de que el módulo web ha de hacer uso, y que son introducidas por los diferentes comandos según va siendo necesario hacerlo. Además, existen otros archivos similares, en el mismo directorio que este, llamados **applicationContext-platform.xml**, **applicationContext-portal.xml** y **applicationContext-security.xml**, que son creados por la ejecución de los comandos Roo++ relativos a la plataforma de la organización, al portal web de la misma, y a la gestión de usuarios, respectivamente, y que incluyen referencias a los beans necesarios para cada una de estas 3 tareas.
- **src/main/tomcat:** contiene los archivos de configuración del despliegue de la aplicación en el servidor web Tomcat: **context.xml** (configuración del acceso a la base de datos cuando la aplicación desarrollada es desplegada en Tomcat) y **server.xml** (configuración general del

servidor web Tomcat para cuando la aplicación desarrollada sea desplegada en él).

- **src/main/webapp/WEB-INF:** contiene el archivo **web.xml** (que configura varios aspectos generales de la aplicación web), y los directorios:
    1. ***classes*** (que contiene un archivo .properties por cada una de las hojas de estilos CSS que se pueden utilizar, definiendo que su hoja de estilos asociada es la que ha de ser usada; en el fichero application.properties es donde se selecciona cuál de estos temas es el que se utilizará).
    2. ***i18n*** (contiene 2 archivos: **application.properties** *(no confundir con el otro archivo del mismo nombre, que es el que en el resto de ocasiones en que aparece este nombre se menciona)* y **messages.properties**, en que se definen las propiedades necesarias para que los elementos de la interfaz web se muestren localizadas en el idioma del usuario).
    3. ***layouts*** (contiene un archivo .jspx por cada layout de la interfaz web que se quiera tener disponible para su uso, y el archivo **layouts.xml**, que define cuál de los layouts se ha de usar en la aplicación).
    4. ***spring*** (contiene un solo archivo, **webmvc-config.xml**, que define el comportamiento modelo-vista-controlador de la aplicación).
    5. ***tags*** (contiene, en una serie de subdirectorios, diferentes archivos .tagx con los elementos a mostrar en la interfaz web, incluyendo menús, formularios, etc).
    6. ***views*** (contiene un archivo .jspx por cada una de las vistas diferentes que ha de ser capaz de mostrar la aplicación web, además de un archivo **views.xml** que define en qué ocasiones se han de mostrar algunas de ellas; algunas de estas vistas son introducidas por los diferentes comandos de Roo++ según van siendo necesarias).
- **src/main/webapp:** además del anteriormente descrito, contiene los siguientes subdirectorios: ***images*** (imágenes a mostrar en la interfaz web), ***scripts*** (archivos JavaScript que han de formar parte de la interfaz web), y ***styles*** (hojas de estilos CSS que definen la apariencia de la aplicación web).

2. Ficheros de configuración

Todos los archivos de configuración de Roo++ se encuentran situados en un directorio llamado *RooAddonsData*, ubicado en el directorio raíz del usuario que está usando la aplicación, tal como *C:\Users\Usuario* o */home/usuario*, o bien, alternativamente, dicho directorio de configuración podrá encontrarse, a elección del usuario o administrador, en cualquier otra ubicación, permitiendo de esta forma la posibilidad su uso por múltiples usuarios dentro de una misma máquina. Para usar esta última opción, ha de definirse una variable de entorno en el sistema operativo, con el nombre *ROOADDON (%ROOADDON%* en Windows o *$ROOADDON* en Unix/Linux), y cuyo contenido sea la ruta completa de la carpeta (incluyendo su nombre, que en este caso podría ser uno diferente a *RooAddonsData*), acabado en el carácter */*. Si esta variable de entorno no está definida, Roo++ asume que el directorio es *RooAddonsData* en el directorio *home* del usuario, y si éste no existe o no tiene el contenido que debe no podrá funcionar adecuadamente.
Este directorio de configuración contiene una serie de subdirectorios; la existencia de alguno de ellos (y de alguno de los ficheros) es opcional, y su presencia o ausencia cambiará el comportamiento de Roo++ ante determinados comandos. Veamos a continuación cuáles son los ficheros y directorios de configuración, y cuál es el cometido y el formato de cada uno de ellos (asumiendo que *RooAddonsData* es el nombre del directorio raíz de la configuración; sustituirlo por el nombre correspondiente en caso de que se utilice otro). Todos los ficheros y directorios se

muestran por orden alfabético. Los nombres (o partes del nombre) que son variables aparecen en *cursiva*. Cuando se hace referencia a un tipo de fichero que puede encontrarse, en su directorio correspondiente, ninguna, una, o varias veces, su nombre aparecerá entre [corchetes]. Si la existencia de un directorio o fichero es opcional, su nombre aparecerá entre (paréntesis).
En la descripción de toda la estructura de ficheros de configuración se asume que el directorio en que se encuentran se llama *RooAddonsData*.
NOTA: *el directorio "RooAddonsData/layout" será tratado de forma diferente al resto, por sus características particulares.*

2.1 RooAddonsData

applicationContext-platform.xml

Fichero XML que contiene los beans que sean necesarios para que la aplicación desarrollada pueda hacer uso de una plataforma.

applicationContext-portal.xml

Fichero XML que contiene los beans que sean necesarios para que el módulo web de la aplicación desarrollada pueda ser integrado en un portal.

breadcrumb-dependency.xml

Contiene el nodo XML con la dependencia respecto al módulo de la plataforma utilizada que introduce las breadcrumbs o "migas de pan" en la interfaz web de la aplicación desarrollada; por ejemplo:

```
<dependency>
      <groupId>com.empresa.platform</groupId>
      <artifactId>platform-readcrumb-module</artifactId>
      <version>${platform.version}</version>
</dependency>
```

breadcrumbs-beans.xml

Contiene uno o varios nodos XML, cada uno de los cuales corresponde a un bean necesario para introducir las "migas de pan" en el módulo web de la aplicación desarrollada.
Cada *bean* debe tener nodos diferentes de apertura y cierre, incluso aunque no tenga nodos hijos; esto es necesario para que Roo++ pueda introducir ciertas propiedades en los mismos.
No ha de ser necesariamente un archivo XML válido en sí mismo (si contiene varios beans, no tendrá nodo raíz).

breadcrumbs-url.txt

Indica la URL a introducir en la cabecera *de src/main/webapp/WEB-INF/layouts/default.jspx*, dentro del módulo web del proyecto en desarrollo, para el uso correcto de las las breadcrumbs o "migas de pan".

context-dataload.sql

Contiene la lista de comandos SQL que se han de ejecutar para introducir en ciertas tablas de la base de datos asociada al proyecto todo lo necesario para el uso de los contextos.

context-dependencies.txt

Contiene, en líneas diferentes, el valor de los campos *groupId*, *artifactId* y *version*, respectivamente, de cada dependencia que se ha de añadir al archivo *pom.xml* del módulo *web* del proyecto para el uso de los contextos (repitiéndose una secuencia de estas 3 líneas para cada una de

las dependencias que se han de introducir).

context-messages.properties

Contiene los mensajes de localización necesarios para el uso de los contextos.

context-portal.properties

Contiene las propiedades que es necesario aplicar al portal utilizado para permitir el uso de los contextos.

licenses.xml

Contiene un nodo XML que define la/s licencia/s que se quiere/n aplicar a la aplicación en desarrollo.
Su contenido ha de ser similar a:

```
<licenses>
  <license>
    <name>Software license</name>
    <url>${license.url}softwarelicense/license.txt</url>
    <distribution>repo</distribution>
    <comments>This is a software license</comments>
  </license>
</licenses>
```

locations-application-properties.xml

Contiene una serie de nodos XML (uno en cada línea), que serán introducidos en el bean *applicationPropertiesBean*, dentro del *pom.xml* del módulo web del proyecto que se está desarrollando. Estos nodos indican ubicaciones adicionales en que se han de buscar los ficheros *.properties* a tener en cuenta durante la construcción del proyecto (además de las que ya incluye el bean tal y como es introducido por defecto). Las ubicaciones configuradas tendrán siempre mayor prioridad sobre las demás que las incluidas por defecto, y cuanto más abajo aparezcan en *locations-application-properties.xml*, mayor prioridad tendrán.
Este fichero no es un fichero XML válido por sí mismo, pues no tiene nodo raíz; si no es necesario introducir nuevos nodos de este tipo en el bean, el fichero debe estar vacío.
Un ejemplo del posible contenido de este archivo es:

```
<value>file:///${TEST_HOME}/test-core.properties</value>
<value>file:///${TEST_HOME}/test-web.properties</value>
```

messages.properties

Indica el texto que se mostrará en la interfaz web de la aplicación desarrollada, cuando ésta haga uso de un portal. Su contenido ha de ser el siguiente, reemplazando los nombres por los que sean necesarios:

```
application_name=Aplicación

#Informacion de los modulos de la aplicacion en el menu general
application.name=Nombre de la Aplicación
application.description=Descripción de la aplicación


# Modulo de configuracion de la aplicacion
application.configuration.label.name=Administración
application.configuration.label.description=Menú de administración


#Casfailed
label.authorization.failure.title=Error de autenticación
```

```
label.authorization.failure.access=Error de acceso
label.global.volver=Volver
label.global.salir=Salir
label.global.footer=Texto del footer

portal.configuration.default.module.name=Administrar
portal.configuration.default.module.description=Módulo de administración
```

mssql.txt

Contiene 2 líneas, indicando, respectivamente, la versión del driver JTDS y la versión del dialecto SQL. Este fichero tan sólo será utilizado cuando la aplicación desarrollada haga uso de una base de datos de Microsoft SQL Server.

mysql.txt

Contiene 2 líneas, indicando, respectivamente, la versión del driver de MySQL para Java, y la versión del dialecto SQL. Este fichero tan sólo será utilizado cuando la aplicación desarrollada haga uso de una base de datos de MySQL.

oracle_sid.txt

SID a utilizar cuando la aplicación desarrollada hace uso de una base de datos Oracle. En otro caso, el contenido de este archivo no será utilizado.

oracle.txt

Contiene 2 líneas, indicando, respectivamente, la versión del driver de bases de datos Oracle para Java, y la versión del dialecto SQL. Este fichero tan sólo será utilizado cuando la aplicación desarrollada haga uso de una base de datos de Oracle.

(parent.txt)

Contiene 3 líneas, que indican, respectivamente, los campos *groupId*, *artifactId* y *version* del módulo que actuará como padre de la aplicación desarrollada.
Si este fichero no existe o no tiene el contenido completo, el proyecto desarrollado carecerá de padre.

platform.txt

Incluye, respectivamente, en diferentes líneas, los campos *groupId* y *version,* y el nombre del módulo que contiene clases de utilidades para llevar a cabo los test de la aplicación desarrollada, correspondientes a la plataforma en la que se integrará la aplicación.

plugin-license.xml

Contiene un plugin para Maven que será añadido al fichero *pom.xml* raíz del proyecto en desarrollo, y que establece como obtener y procesar la licencia de dicho proyecto.
Su contenido ha de ser similar a:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>license-maven-plugin</artifactId>
  <version>1.0</version>
    <configuration>
      <licenseName>license</licenseName>
      <licenseResolver>${license.url}</licenseResolver>
      <bundleLicensePath>META-INF/LICENSE.txt</bundleLicensePath>
      <generateBundle>false</generateBundle>
    </configuration>
```

```
    <executions>
      <execution>
        <id>update-project-license</id>
        <goals>
          <goal>update-project-license</goal>
        </goals>
        <phase>process-sources</phase>
      </execution>
  </executions>
</plugin>
```

portal.properties

Propiedades necesarias para integrar en un portal el módulo web de la aplicación que se está desarrollando. Su contenido ha de ser similar al siguiente, aunque puede variar dependiendo de las características concretas del portal utilizado:

```
# Beans que proporcionan la información de la aplicación
portal.application.provider.name=portalApplicationProvider
portal.configuration.provider.name=portalConfigurationProvider

# Properties de configuración del portal
portal.application.id=${APP.NAME}
portal.application.url=${REMOTE.HOST}/${APP.NAME}
portal.application.priority=1
portal.application.default.name=${APP.NAME}
portal.application.default.description=${APP.NAME}

portal.default.module.id=Default
portal.default.module.url=${portal.application.url}/index

portal.configuration.logout.url=/j_spring_security_logout
portal.configuration.user.url=${portal.application.url}/userInfo
portal.configuration.security.role=ROLE_ADMIN

portal.configuration.default.module.id=Configuration
portal.configuration.default.module.url=${portal.application.url}/index
portal.configuration.default.module.role=ROLE_ADMIN

portal.default.module.default.name=Default
portal.default.module.default.description=Default
```

portal.txt

Contiene los campos *groupId*, *artifactId* y *version* de la dependencia del proyecto desarrollado con respecto al módulo correspondiente al portal en el que se ha de integrar su módulo web.

portal-header.jspx

Archivo JavaServer Pages (*.jspx*) que introduce en la interfaz de la aplicación web desarrollada la cabecera correspondiente al portal en que ésta se integra.
Su contenido puede ser algo similar a:

```
<div id="header" xmlns:jsp="http://java.sun.com/JSP/Page"
       xmlns:fn="http://java.sun.com/jsp/jstl/functions"
       xmlns:c="http://java.sun.com/jsp/jstl/core"
       xmlns:spring="http://www.springframework.org/tags"
       xmlns:portal="http://www.empresa.com/portal" version="2.0">

      <jsp:directive.page
import="org.springframework.web.servlet.support.RequestContextUtils" />
      <jsp:directive.page
import="org.springframework.context.i18n.LocaleContextHolder" />
```

```
      <jsp:directive.page
import="com.empresa.portal.handler.PortalClientHandler" />

      <jsp:scriptlet>
            pageContext.setAttribute("applications",
PortalClientHandler.getApplications(), PageContext.PAGE_SCOPE);
        pageContext.setAttribute("configuration",
PortalClientHandler.getConfiguration(), PageContext.PAGE_SCOPE);
        pageContext.setAttribute("currentLocale",
LocaleContextHolder.getLocale(), PageContext.PAGE_SCOPE);
        pageContext.setAttribute("requestUri",
org.springframework.security.web.util.UrlUtils
            .buildFullRequestUrl(pageContext.getRequest().getScheme(),
pageContext.getRequest().getServerName(),
                  pageContext.getRequest().getServerPort(),
                  pageContext.getAttribute("javax.servlet.forward.request_uri",
2).toString(), null), 2);
      </jsp:scriptlet>

      <portal:portal-header applications="${applications}" currentLocale="$
{currentLocale}"
            configuration="${configuration}" requestUri="${requestUri}" />

</div>
```

portal-url.txt

URL correspondiente al portal a utilizar en el módulo web de la aplicación en desarrollo.

postgresql.txt

Contiene 2 líneas, indicando, respectivamente, la versión del driver de PostgreSQL para Java y la versión del dialecto SQL. Este fichero tan sólo será utilizado cuando la aplicación desarrollada haga uso de una base de datos de PostgreSQL.

usermgmt.txt

Este fichero sólo será usado en caso de que se utilicen las librerías de Spring Security para la gestión de usuarios, en lugar de un módulo personalizado. Contiene 6 líneas, cuyo contenido es, por orden:

- Nombre de la máquina en la que se ejecutará la aplicación en desarrollo.
- Puerto en el que se accederá a la interfaz web de la aplicación en desarrollo (típicamente será 8080, aunque puede ser otro si así se establece en el servidor web utilizado).
- URL de la página de autenticación (*login*) del sistema CAS (Central Authentication Service) utilizado, incluyendo el número de puerto.
- URL del sistema CAS que valida el *ticket* (indicando que el usuario se haya autenticado correctamente y que su sesión no haya caducado), incluyendo el número de puerto (que coincide con la URL base del CAS).
- Un identificador único para el proveedor de autenticación (puede tener cualquier valor, tal como el nombre de la aplicación o de la organización).
- URL de la página de *logout* del sistema CAS, incluyendo el número de puerto.

usermgmt_application.properties

Todas las propiedades que sea necesario añadir al archivo de este mismo nombre, dentro del módulo web de la aplicación en desarrollo, para llevar a cabo la gestión de usuarios desde la propia aplicación, mediante las librerías de Spring Security. Si dicha gestión de usuarios se realiza a través

de un módulo externo, configurado mediante el directorio *usermgmt*, este archivo no será utilizado. Por ejemplo, su contenido podría ser algo como:

```
# Role security
application.security.role=ROLE_ADMIN
```

Dependiendo siempre del nivel de protección deseado.

usermgmt_users.txt

Este fichero sólo será usado en caso de que se utilicen las librerías de Spring Security para la gestión de usuarios, en lugar de un módulo personalizado. Contiene 3 líneas por cada uno de los usuarios de la aplicación desarrollada: nombre de usuario, contraseña, y roles del usuario (por ejemplo, *ROLE_USER* o *ROLE_USER, ROLE_ADMIN*). Las líneas han de encontrarse en este orden, y a continuación de ellas las mismas 3 líneas correspondientes al siguiente usuario de la lista, si es que existe. En este fichero deben figurar estos 3 atributos para cada uno de los usuarios que deseen acceder a la aplicación protegida, y sus nombres de usuario, contraseñas y roles han de ser los mismos que existan en el sistema CAS (Central Authentication Service) utilizado para la autenticación.

2.2 RooAddonsData/contextBeans/portal

[*portal_bean.*xml]

Archivos que contienen, cada uno de ellos, un nodo XML, referente a un bean que es necesario introducir en el archivo applicationContext-portal.xml (no en el archivo de configuración, si no en su copia creada en el proyecto tras ejecutar el comando portal setup) para el uso de los contextos.

2.3 RooAddonsData/contextBeans/portal/nodesToAdd

[*node_to_add.*txt]

Cada uno de estos archivos contiene los datos sobre un nodo XML a introducir en el archivo applicationContext-portal.xml (no en el archivo de configuración, si no en su copia creada en el proyecto tras ejecutar el comando portal setup), con el fin de permitir el uso de los contextos. Ha de contener, en líneas diferentes, respectivamente, el XPath del nodo al que se le introducirá el nuevo elemento como hijo, el nombre del nuevo nodo a introducir como hijo de éste, y los diferentes atributos, cada uno de ellos seguido, en la siguiente línea, por su correspondiente valor.

2.4 RooAddonsData/contextBeans/security

authenticationSuccessHandler.txt

Contiene el nombre del bean introducido en el archivo applicationContext-security.xml (situado en el módulo web del proyecto tras la ejecución del comando usermgmt setup) que será el encargado de gestionar la situación de éxito en la autenticación de usuarios, en lugar del que lo hace por defecto (*authenticationSuccessHandler*), para posibilitar el uso de los contextos.

[*security_bean.*xml]

Archivos que contienen, cada uno de ellos, un nodo XML, referente a un bean que es necesario introducir en el archivo applicationContext-security.xml (situado en el módulo web del proyecto tras la ejecución del comando usermgmt setup) para hacer posible el uso de los contextos.

2.5 RooAddonsData/contextHeader

add-to-portal-header.txt

Contiene las asignaciones de variables que sea necesario añadir al nodo *portal:portal-header*, en el archivo *src/main/webapp/WEB-INF/views/header.jspx* (dentro del módulo web del proyecto), para hacer posible el uso de contextos por parte de la aplicación.

directive-pages.jspx

Contiene los nodos "*<jsp:directive.page>*"que se han de introducir en el archivo *src/main/webapp/WEB-INF/views/header.jspx* (dentro del módulo web del proyecto) para posibilitar el uso de los contextos. No tiene por qué ser un archivo jspx válido (puede no tener nodo raíz).

scriptlets.jspx

Contiene los nodos "*<jsp:scriptlet>*"que se han de introducir en el archivo *src/main/webapp/WEB-INF/views/header.jspx* (dentro del módulo web del proyecto) para posibilitar el uso de los contextos. No tiene por qué ser un archivo jspx válido (puede no tener nodo raíz).

2.6 RooAddonsData/contextViews

[*vista*.jspx]

Cada uno de estos archivos contiene una vista diferente, necesaria para la visualización de la aplicación según los contextos utilizados.

2.7 RooAddonsData/doc

fonts.xml

Lista de fuentes que serán utilizadas en la documentación del proyecto en desarrollo, incluyendo la ubicación de sus archivos correspondientes. Una misma fuente puede aparecer en varias ocasiones, cuando se hace uso de la misma con varios estilos diferentes (negrita, cursiva, etc).
Dicha lista debe formar parte de un nodo XML de nombre *fonts*. Un ejemplo de posible contenido para este archivo sería:

```
<fonts>
      <font>
            <name>Calibri</name>
            <style>normal</style>
            <weight>normal</weight>
            <embedFile>${basedir}/src/fonts/calibri.ttf</embedFile>
            <metricsFile>${project.build.directory}/fonts/calibri-
metrics.xml</metricsFile>
      </font>
      <font>
            <name>Calibri</name>
            <style>italic</style>
            <weight>normal</weight>
            <embedFile>${basedir}/src/fonts/calibrii.ttf</embedFile>
            <metricsFile>${project.build.directory}/fonts/calibrii-
metrics.xml</metricsFile>
      </font>
      <font>
            <name>Calibri</name>
            <style>normal</style>
            <weight>bold</weight>
```

```
            <embedFile>${basedir}/src/fonts/calibrib.ttf</embedFile>
            <metricsFile>${project.build.directory}/fonts/calibrib-
metrics.xml</metricsFile>
      </font>
      <font>
            <name>Georgia</name>
            <style>normal</style>
            <weight>normal</weight>
            <embedFile>${basedir}/src/fonts/georgia.ttf</embedFile>
            <metricsFile>${project.build.directory}/fonts/georgia-
metrics.xml</metricsFile>
      </font>
</fonts>
```

En este ejemplo , *${basedir} /src/fonts/georgia.ttf* le indica a Maven que localice el archivo que contiene esa fuente en esa ubicación, dentro del módulo *docs* (a cuyo directorio raíz se refiere *${basedir}*, ya que este nodo *fonts* será introducido en el fichero *pom.xml* de dicho módulo), mientras que *${project.build.directory}/fonts/georgia-metrics.xml* le indica que el archivo de métricas de dicha fuente se generará en ese lugar dentro del directorio *target* (el directorio en que se guardan los resultados durante la compilación del proyecto) de dicho módulo *docs*.

2.8 RooAddonsData/doc/docbkx

custom.xsl

Hoja de estilos XSL que se utilizará cuando se genere la documentación del proyecto en desarrollo mediante DocBook XSL, para determinar la presentación de la misma.

2.9 RooAddonsData/doc/fonts

[*fontname.ttf*]

Archivos conteniendo las fuentes (TrueType, OpenType, o cualquier otro tipo) que serán utilizadas en la documentación del proyecto en desarrollo.

2.10 RooAddonsData/doc/front_pages

(Installation_Manual.pdf)

Archivo PDF que contiene una serie de páginas prefijadas (probablemente portadas , contraportadas, u otras páginas especiales que no se vayan a introducir mediante el fichero XML de DocBook XSL *src/InstallationManual.xml* (dentro del módulo *docs* del proyecto).
Las reglas para su introducción en el documento *Installation_Manual.pdf* (manual de implantación de la aplicación desarrollada) final están en RooAddonsData/docPlugins.
Si este archivo no está presente, RooAddonsData/docPlugins debe estar vacío, o los plug-ins allí contenidos no obtener ninguna página del archivo *front_pages/Installation_Manual.pdf*.

(User_Manual.pdf)

Archivo PDF que contiene una serie de páginas prefijadas (probablemente portadas, contraportadas, u otras páginas especiales que no se vayan a introducir mediante los ficheros XML de DocBook XSL *src/UserManual.xml* (dentro del módulo *docs* del proyecto).
Las reglas para su introducción en el documento *User_Manual.pdf* (manual de implantación de la aplicación desarrollada) final están en RooAddonsData/docPlugins.
Si este archivo no está presente, RooAddonsData/docPlugins debe estar vacío, o los plug-ins allí contenidos no obtener ninguna página del archivo *front_pages/User_Manual.pdf*.

2.11 RooAddonsData/docPlugins

[*pluginname*.xml]

Ninguno, uno, o varios ficheros XML conteniendo plug-ins de Maven que serán incluidos en el fichero *pom.xml* correspondiente al módulo *docs* del proyecto. La función de estos plug-ins será determinar el modo en que se genere la documentación del proyecto en desarrollo a partir de lo que se haya colocado en los archivos de configuración custom.xsl, Installation_Manual.pdf, User_Manual.pdf y fonts.xml, y de lo que, una vez se haya generado el proyecto, se haya introducido en los archivos *src/InstallationManual.xml* y *src/UserManual.xml*, dentro del módulo *docs* del proyecto. El objetivo de este/estos plug-in/s es el de generar los correspondientes *Installation_Manual.pdf* y *User_Manual.pdf*, que contendrán la documentación final del software en desarrollo orientada al implantador y al usuario, respectivamente. Si no se incluye ningún plug-in, los dos manuales serán solamente los documentos PDF generados por DocBook XSL a partir de *src/InstallationManual.xml* y *src/UserManual.xml*.
La estructura de un fichero XML que contenga un plug-in básico para este propósito ha de ser la siguiente:

```
<plugin>
	<groupId>com.organizacion.maven.plugins</groupId>
	<artifactId>maven-documentation-plugin</artifactId>
	<version>1.0</version>
	<configuration>
		<builds>
			<BuildDocument>
				<inputFiles>
					<PdfConfiguration>
	<file>src/front_pages/Installation_Manual.pdf</file>
	<pagesPattern>1-2</pagesPattern>
					</PdfConfiguration>
					<PdfConfiguration>
						<file>target/help-
pdf/InstallationManual.pdf</file>
						<pagesPattern>3-*</pagesPattern>
					</PdfConfiguration>
					<PdfConfiguration>

<file>src/front_pages/
Installation_Manual.pdf</file>
	<pagesPattern>3-3</pagesPattern>
					</PdfConfiguration>
				</inputFiles>
				<outputFile>target/Installation_Manual.pdf</outputFile>
			</BuildDocument>
			<BuildDocument>
				<inputFiles>
					<PdfConfiguration>
						<file>src/front_pages/User_Manual.pdf</file>
						<pagesPattern>1-2</pagesPattern>
					</PdfConfiguration>
					<PdfConfiguration>
						<file>target/help-pdf/UserManual.pdf</file>
						<pagesPattern>3-*</pagesPattern>
					</PdfConfiguration>
                                                        <PdfConfiguration>
					<file>src/front_pages/User_Manual.pdf</file>
					<pagesPattern>3-3</pagesPattern>
					</PdfConfiguration>
				</inputFiles>
				<outputFile>target/User_Manual.pdf</outputFile>
			</BuildDocument>
		</builds>
	</configuration>
	<executions>
```

```
            <execution>
                  <id>generate-full-am-help</id>
                  <phase>generate-resources</phase>
                  <goals>
                        <goal>compose-pdf</goal>
                  </goals>
            </execution>
      </executions>
</plugin>
```

Este ejemplo asume que los dos archivos *Installation_Manual.pdf* y *User_Manual.pdf*, situados en *RooAddonsData/doc/front_pages*, tienen ambos 3 páginas, correspondientes, respectivamente, a las 2 primeras y a la última de los documentos finales de sus mismos nombres que se generarán , siendo el resto de su contenido creado automáticamente a partir de lo indicado en los ficheros de configuración *fonts.xml* y *custom.xsl*, y del texto incluido en los archivos *src/InstallationManual.xml* y *src/UserManual.xml*, dentro del módulo *docs* del proyecto en desarrollo.

2.12 RooAddonsData/layout

Directorio que contiene diversos elementos que serán copiados al módulo web del proyecto en desarrollo (concretamente, dentro de *src/main/webapp/WEB-INF*) para que éste adquiera el *layout* (apariencia y distribución de los elementos en la interfaz) deseado. Su contenido se encuentra estructurado en una serie de subdirectorios:

- **classes:** archivos *.properties* que seleccionar estilos diferentes para modificar la apariencia de la interfaz web de la aplicación. Un ejemplo de su posible contenido es: `styleSheet=resources/styles/hojaDeEstilos.css`
- **images:** archivos de imágenes (JPEG, PNG, etc.).
- **layouts:** archivos JavaServer Pages (*.jspx*), cada uno de los cuales contiene un modo de distribuir los diferentes elementos en la página web.
- **properties:** archivos *.properties* que serán introducidos en el directorio de *i18n*, que indican diferentes propiedades relacionadas con la internacionalización y localización de la aplicación.
- **scripts:** archivos JavaScript (*.js*) que se desea se ejecuten dentro de la aplicación web desarrollada.
- **styles:** archivos que contienen Cascading Style Sheets (*.css*) que determinarán la apariencia de la interfaz de la aplicación web.
- **tags:** archivos de tags de JavaServer Pages (*.tagx*), que contienen definiciones de elementos a mostrar en la interfaz web. Habitualmente se encontrarán a su vez en subdirectorios, dentro de *tags*, según el tipo de elemento (menú, formulario, etc.) de que se trate. Ver los archivos de este creados automáticamente junto con el módulo web para obtener más información.

Además, en *RooAddonsData/layout* se encuentra el fichero **load-scripts.jspx**, que ha de contener, por cada JavaScript que haya en el directorio *scripts*:

```
<spring:url value="/resources/scripts/script.js" var="script_url" />
<script src="${script_url}" type="text/javascript"><!-- required for FF3 and
Opera --></script>
```

Y por cada CSS que haya en el directorio *styles*:

```
<spring:url value="/resources/styles/estilo.css" var="estilo_url" />
<link rel="stylesheet" type="text/css" href="${estilo_url}" />
```

(No se trata de un archivo JSPX (JavaServer Pages) completo, tan sólo de estas líneas que serán introducidas en el archivo del mismo nombre, dentro del proyecto en desarrollo).

2.13 (RooAddonsData/usermgmt)

Este directorio sólo ha de existir en caso de que se desee hacer uso de un módulo externo de gestión de usuarios, y nunca si lo que se quiere es llevar a cabo la gestión de usuarios directamente desde la propia aplicación en desarrollo, mediante Spring Security y CAS (Central Authentication Service). Su función es configurar cómo se ha de llevar a cabo la gestión de usuarios mediante el módulo externo. La existencia o no existencia de este directorio determinará el tipo de gestión de usuarios que se llevará a cabo cuando se ejecute el comando Roo++ correspondiente.

application.properties

Todas las propiedades que sea necesario añadir al archivo de este mismo nombre, dentro del módulo web de la aplicación en desarrollo, para hacer uso del módulo de gestión de usuarios. Por ejemplo, su contenido podría ser algo como:

```
# Remote security
AUTHENTICATION.REQUIRED=true
# Autenticacion para servicios remotos
REMOTE.USER.USERNAME=user
REMOTE.USER.PASSWORD=password
PASSWORD.ENCRYPT.TYPE=SIMPLE
# Properties file with server URL settings for remote access
cas.securityContext.service.url=${REMOTE.HOST}/${APP.NAME}
cas.securityContext.serviceProperties.service=${REMOTE.HOST}/$
{APP.NAME}/j_spring_cas_security_check
cas.securityContext.ticketValidator.casServerUrlPrefix=${REMOTE.USERMODULE.HOST}/$
{USERMGMT.NAME}
cas.securityContext.casAuthenticationFilterEntryPoint.loginUrl=$
{cas.securityContext.ticketValidator.casServerUrlPrefix}/login
cas.securityContext.casAuthenticationFilterEntryPoint.logoutUrl=$
{cas.securityContext.ticketValidator.casServerUrlPrefix}/j_spring_security_logout
cas.securityContext.remoteUsersService=${REMOTE.USERMODULE.HOST}/$
{USERMGMT.NAME}/remoting/remoteUsersService
cas.securityContext.remotePrincipalService=/$
{USERMGMT.NAME}/remoting/remotePrincipalService
cas.securityContext.remoteUserDetailsService=${REMOTE.USERMODULE.HOST}/$
{USERMGMT.NAME}/remoting/remoteUserDetailsService
cas.securityContext.remoteAuthenticationManagerService=${REMOTE.USERMODULE.HOST}/$
{USERMGMT.NAME}/remoting/remoteAuthenticationService
# Role security
application.security.role=ROLE_ADMIN
```

Este contenido dependerá de aspectos tales como el nivel de seguridad que se le quiera dar a la aplicación, o las características particulares que tenga el módulo de gestión de usuarios utilizado.

applicationContext-security.xml

Fichero XML que contiene los beans que sean necesarios para que el módulo web de la aplicación desarrollada utilice la gestión de usuarios proporcionada por un módulo externo.

artifcactItems.txt

Contiene los diferentes elementos artifactItem que han de ser introducidos en el fichero *pom.xml* del módulo *package* del proyecto, con el fin de empaquetar diferentes elementos relativos al de gestión de usuarios, cuando ésta se hace a través de un módulo externo en lugar de con las librerías de Spring Security.
Su contenido ha de ser similar al siguiente:

```
usermgmt-application
war
${usermgmt.delivery}.war
usermgmt-doc
zip
${usermgmt.delivery}-docs.zip
```

```
usermgmt-scripts
zip
${usermgmt.delivery}-scripts.zip
```
En este ejemplo vemos que hay 3 *artifactItems* diferentes, y para cada uno de ellos se indica en una línea diferente el valor de los campos *artifactId*, type y destFileName.

changePassword.jspx

Archivo JavaServer Pages (*.jspx*) que introduce en la interfaz web de la aplicación en desarrollo la posibilidad permitir usuario que la está usando cambiar su contraseña.
Su contenido podría ser similar a:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<div id="user_password" xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:userInfo="http://www.empresa.com/usermgmt/userInfo"
      xmlns:spring="http://www.springframework.org/tags"
      class="body_info"
      version="2.0">
      <jsp:output omit-xml-declaration="yes" />

            <fieldset>
                  <legend>
                        <spring:message
                              code="label_usermgmt_user_username"
                              arguments="${userPasswordChange.username}" />
                  </legend>
                  <userInfo:changePassword />
            </fieldset>

</div>
```

userInfo.jpsx

Archivo JavaServer Pages (*.jspx*) que introduce en la interfaz web de la aplicación en desarrollo la posibilidad de mostrar una página con el perfil del usuario que la está usando en cada momento.
Su contenido podría ser similar a:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<div id="user_info" xmlns:jsp="http://java.sun.com/JSP/Page"
      xmlns:userInfo="http://www.empresa.com/usermgmt/userInfo"
      xmlns:spring="http://www.springframework.org/tags"
      class="body_info"
      version="2.0">
      <jsp:output omit-xml-declaration="yes" />


            <fieldset>
                  <legend>
                        <spring:message
                              code="label_usermgmt_user_username"
                              arguments="${user.username}" />
                  </legend>
                  <userInfo:userInfo />
            </fieldset>
</div>
```

usermgmt.txt

Contiene 5 líneas, que definen, respectivamente, los campos *groupId*, *artifactId* y *version* del módulo de gestión de usuarios, la URL completa (incluyendo en la misma el número de puerto correspondiente) en la que se encuentra dicho módulo, y el nombre de la aplicación de gestión de usuarios que este módulo proporciona (este último valor será asignado a la variable `$`

```
{usermgmt.delivery}).
```
Su contenido ha de ser algo tal como:
```
com.empresa.usermgmt
usermgmt-client
1.0
http://maquina.empresa.com:8080
appname
```

3. "Resources" del JAR

El proyecto Maven de Roo++ incluye 2 directorios fuente dentro del directorio "resources", que por tanto serán introducidos en el JAR final tras su compilación.

Uno de ellos, de nombre RooAddonsData, contiene los diferentes archivos del directorio de configuración que acabamos de abordar, con una serie de valores por defecto, de ejemplo, simplemente con el fin de que esta estructura de archivos de configuración se encuentre integrada de alguna forma en el directorio fuente del proyecto.

El otro directorio de "resources", el único que es usado como tal por el software, tiene como nombre "rooAddon", y está formado por los ficheros que veremos a continuación.

Los archivos de los directorios *core*, *docs*, *ear*, *package* y *scripts*, además del pom.xml situado en la raíz del directorio, son copiados a la misma ubicación dentro de cada proyecto en desarrollo durante la ejecución del comando `roo++ project create-modules`; en el caso de los situados en el directorio *web*, son copiados al proyecto en desarrollo durante la ejecución de otros de los comandos de Roo++, o, en ocasiones, introducidos dentro de otros archivos, mientras que los archivos dirs.txt y textfiles.txt indican, respectivamente, qué directorios se han de crear y qué archivos de texto se han de copiar en la ejecución de `roo++ project create-modules`.

Es importante tener en cuenta que muchos de estos archivos contienen valores de ejemplo que son reemplazados por sus valores reales, ajustados a la aplicación que se está desarrollando, durante la ejecución del mismo comando Roo++ que los copia al proyecto.

3.1 core/pom.xml

Archivo pom.xml del módulo core de la aplicación.

3.2 docs/src/assembly/package.xml

Definición de aspectos relacionados cn el empaquetamiento de la documentación durante la compilación del proyecto.

3.3 docs/src/InstallationManual.xml

Estructura básica para la creación, en DocBook, del manual de instalación de la aplicación en desarrollo.

3.4 docs/src/UserManual.xml

Estructura básica para la creación, en DocBook, del manual de usuario de la aplicación en desarrollo.

3.5 docs/pom.xml

Archivo pom.xml del módulo docs de la aplicación.

3.6 ear/src/main/application/META-INF/weblogic-application.xml

Preferencias para el empaquetamiento de la aplicación desarrollada en un archivo EAR.

3.7 ear/pom.xml

Archivo pom.xml del módulo ear de la aplicación.

3.8 package/src/assembly/package.xml

Configuración del empaquetamiento final del proyecto, como último paso de su compilación.

3.9 package/pom.xml

Archivo pom.xml del módulo package de la aplicación, en el que se configuran los tipos de archivo y el modo en que ésta se ha de empaquetar tras su compilación.

3.10 scripts/src/assembly/compress-zip.xml

Definición del formato ZIP como forma de agrupar y comprimir el contenido del módulo scripts durante la compilación del proyecto.

3.11 scripts/pom.xml

Archivo pom.xml del módulo scripts de la aplicación.

3.12 web/src/main/jetty/etc/realm.properties

Definición de usuarios y contraseñas para el uso de Jetty.

3.13 web/src/main/jetty/etc/webdefault.xml

Configuración de Jetty común a todas las aplicaciones.

3.14 web/src/main/jetty/jetty-database.xml

Contiene el nodo XML que define el uso de una base de datos desde una aplicación web desplegada en Jetty, tras la sustitución de los valores de ejemplo por los reales.

3.15 web/src/main/jetty/jetty.xml

Archivo de configuración de Jetty.

3.16 web/src/main/resources/META-INF/spring/application-properties-bean.xml

Definición de la lista de ubicaciones, y de su prioridad, en que se buscarán los archivos .properties.

3.17 web/src/main/resources/META-INF/spring/application.properties

Propiedades de la aplicación, que serán introducidas en los proyectos desarrollados con Roo++, tras ajustarlas a las características de cada uno de ellos.

3.18 web/src/main/resources/META-INF/spring/applicationContext-security.xml

Beans relacionados con la autenticación de usuarios, cuando ésta se realiza a través de las librerías de Spring Security.

3.19 web/src/main/resources/META-INF/spring/database.properties

Definición de diferentes propiedades relacionadas con la conexión de la aplicación desarrollada a una base de datos, mediante la librería Hibernate.

3.20 web/src/main/resources/META-INF/spring/entity-manager-factory.xml

Definición de diferentes propiedades para la librería Hibernate, encargada de la gestión de las conexiones con la base de datos.

3.21 web/src/main/resources/META-INF/spring/jndi-datasource.xml

Nodo XML dataSource.

3.22 web/src/main/resources/META-INF/spring/message-source.xml

Bean que define la ubicación de archivos de localización.

3.23 web/src/main/resources/META-INF/spring/property-placeholder-configurer.xml

Nodo XML property-placeholder.

3.24 web/src/main/tomcat/context.xml

Configuración del acceso a la base de datos cuando la aplicación desarrollada es desplegada en Tomcat.

3.25 web/src/main/tomcat/server.xml

Configuración general del servidor web Tomcat para cuando la aplicación desarrollada sea desplegada en él.

3.26 web/src/main/webapp/WEB-INF/views/casfailure.jspx

Vista que se mostrará cuando un usuario autenticado trate de acceder a la aplicación sin tener permisos suficientes para ello.

3.27 web/src/main/webapp/WEB-INF/casfailure-view.xml

Referencia al archivo anterior.

3.28 web/plugin-license.xml

Plugin que incluye referencias a 3 licencias bajo las cuales son distribuidas las librerías de Spring.

3.29 web/plugin-source.xml

Plug-in de Maven para la creación de un JAR que incluya el código fuente del proyecto.

3.30 web/resource-ref.xml

Configura el DataSource usado por la aplicación, con el fin de permitir la conexión entre ésta y su base de datos asociada.

3.31 dirs.txt

Lista de directorios creados durante la ejecución del comando `roo++ project create-modules`:

```
core
core/src
core/src/main
core/src/main/java
core/src/main/resources
core/src/test
core/src/test/java
core/src/test/resources
docs
docs/src
docs/src/assembly
docs/src/docbkx
docs/src/fonts
docs/src/front_pages
docs/src/images
ear
ear/src
ear/src/main
ear/src/main/application
ear/src/main/application/META-INF
package
package/src
package/src/assembly
scripts
scripts/src
scripts/src/assembly
scripts/src/main
scripts/src/main/resources
```

3.32 pom.xml

Archivo pom.xml de un proyecto Roo++ completo, incluyendo la lista de los diferentes módulos que lo componen, y el orden en que éstos se han de compilar: core, docs, scripts, web, ear, package. Además, incluye los campos que hacen referencia al parent del proyecto, que si finalmente no existe serán eliminados.

3.33 textfiles.txt

Lista de archivos de texto copiados, desde este directorio de "resources" en el JAR de Roo++, al proyecto, durante la ejecución del comando `roo++ project create-modules`:

```
pom.xml
core/pom.xml
docs/pom.xml
docs/src/InstallationManual.xml
docs/src/UserManual.xml
docs/src/assembly/package.xml
ear/pom.xml
ear/src/main/application/META-INF/weblogic-application.xml
package/pom.xml
package/src/assembly/package.xml
scripts/pom.xml
scripts/src/assembly/compress-zip.xml
```