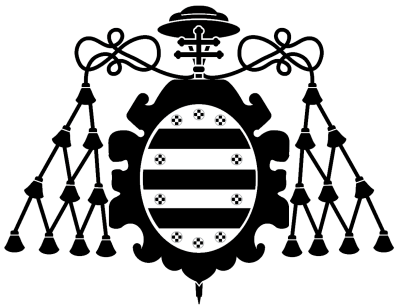


UNIVERSIDAD DE OVIEDO

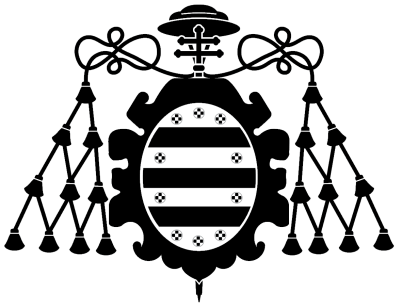
ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

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 VII

CÓDIGO FUENTE

JAVIER MENÉNDEZ ÁLVAREZ
JULIO 2014

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

Índice

Paquete Breadcrumb

BreadcrumbCommands.java

```
package rooplusplus.roo.addon.breadcrumb;

import java.util.logging.Logger;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.osgi.service.component.ComponentContext;
import org.springframework.roo.shell.CliAvailabilityIndicator;
import org.springframework.roo.shell.CliCommand;
import org.springframework.roo.shell.CommandMarker;

/**
 * Comandos relacionados con las "migas de pan" del modulo web de los proyectos.
 * The command class is registered by the Roo shell following an automatic
 * classpath scan. You can provide simple user presentation-related logic in
 * this class. You can return any objects from each method, or use the logger
 * directly if you'd like to emit messages of different severity (and therefore
 * different colors on non-Windows systems).
 *
 * @since 1.1.1
 */
@Component
// Use these Apache Felix annotations to register your commands class in the Roo
// container
@Service
public class BreadcrumbCommands
    implements CommandMarker { // All command types must implement the

    // CommandMarker interface

    /**
     * Constructor de la clase.
     */
    public BreadcrumbCommands() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Get hold of a JDK Logger.
     */
    private static final Logger LOG = Logger
        .getLogger("BreadcrumbCommands.class");

    /**
     * Get a reference to the AddonOperations from the underlying OSGi
     * container.
     */
    @Reference
    private BreadcrumbOperations operations;

    /**
     * The activate method for this OSGi component, this will be called by the
```

```
     * OSGi container upon bundle activation (result of the 'addon install'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void activate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

    /**
     * The deactivate method for this OSGi component, this will be called by the
     * OSGi container upon bundle deactivation (result of the 'addon remove'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void deactivate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

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

        return operations.canBreadcrumbSetup();
    }

    /**
     * Configura las "migas de pan" o breadcrumb para el modulo web del proyecto
     * que tiene el foco.
     */
    @CliCommand(value = "roo++ web breadcrumb setup", help = "Sets up
breadcrumbs for web application")
    public void breadcrumbSetup() {

        operations.breadcrumbSetup();
        LOG.info("Web application breadcrumbs configured");
    }
}
```

BreadcrumbOperations.java

```
package rooplusplus.roo.addon.breadcrumb;


/**
 * Interface of commands that are available via the Roo shell.
 *
```

```
 * @since 1.1.1
 */
public interface BreadcrumbOperations {

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

    /**
     * Configura las "migas de pan" o breadcrumb para el modulo web del proyecto
     * que tiene el foco.
     */
    void breadcrumbSetup();
}
```

BreadcrumbOperationsImp.java

```
package rooplusplus.roo.addon.breadcrumb;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.springframework.roo.project.ProjectOperations;
import org.w3c.dom.Document;
```

```
import org.w3c.dom.Node;
import org.xml.sax.SAXException;

import rooplusplus.roo.addon.utils.FileUtilities;
import rooplusplus.roo.addon.utils.ProjectUtils;
import rooplusplus.roo.addon.utils.WebModuleUtils;
import rooplusplus.roo.addon.utils.XmlUtils;


/**
 * Implementation of {@link BreadcrumbOperations} interface.
 *
 * @since 1.1.1
 */
@Component
@Service
public class BreadcrumbOperationsImpl
    implements BreadcrumbOperations {

    /**
     * Constructor de la clase.
     */
    public BreadcrumbOperationsImpl() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Log para mostrar informacion al usuario.
     */
    private static final Logger LOG = Logger
        .getLogger("BreadcrumbOperationsImpl.class");

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String POM = "pom.xml";

    /**
```

```
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String BREADCRUMB_DEPENDENCY =
        "breadcrumb-dependency.xml";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String XMLNS_UTIL =
        "xmlns:util=\"urn:jsptagdir:/WEB-INF/tags/util\"";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String APP_CONTEXT_PLATFORM =
        "src/main/resources/META-INF/spring/applicationContext-platform.xml";

    /**
     * Get a reference to the ProjectOperations from the underlying OSGi
     * container. Make sure you are referencing the Roo bundle which contains
     * this service in your add-on pom.xml.
     */
    @Reference
    private ProjectOperations projectOperations;

    @Override
    public boolean canBreadcrumbSetup() {

        String webPom =
            FileUtilities.readFileAsString(WebModuleUtils.getWebModuleFile(POM,
                projectOperations).getAbsolutePath());
        File file =
            new File(
                FileUtilities.getConfigurationFilePath(BREADCRUMB_DEPENDENCY));
        if (!file.exists()) {
            return false;
        }
        BufferedReader br = null;
        try {
```

```
            br = new BufferedReader(new FileReader(file));
            br.readLine();
            br.readLine();
            String breadcrumb = br.readLine();
            return ProjectUtils.areModulesCreated(projectOperations)
                && !webPom.contains(breadcrumb)
                && WebModuleUtils.getWebModuleFile(APP_CONTEXT_PLATFORM,
                    projectOperations).exists()
                && WebModuleUtils.isWebJpa(projectOperations);
        } catch (IOException ex) {
            LOG.log(Level.SEVERE,
                "ERROR. Problem reading \"breadcrumb-dependency.xml\"
configuration file");
            return false;
        } finally {
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    @Override
    public void breadcrumbSetup() {

        try {
            String pomPath =
                WebModuleUtils.getWebModuleFile(POM, projectOperations)
                    .getAbsolutePath();
            Document dep =
                XmlUtils.createDocumentFromFile(FileUtilities
                    .getConfigurationFilePath(BREADCRUMB_DEPENDENCY));
            Document doc = XmlUtils.createDocumentFromFile(pomPath);
            Node deps =
                XmlUtils.evaluateXPath("/project/dependencies", doc).item(0);
            deps.appendChild(doc.adoptNode(dep.getDocumentElement()));
            XmlUtils.guardaXml(doc, pomPath);
```

```
            editDefaultLayout();
            addPlatformBeans();
        } catch (IOException ex) {
            breadbrumbDependencyNotFound();
        } catch (ParserConfigurationException e) {
            breadbrumbDependencyNotFound();
        } catch (SAXException e) {
            breadbrumbDependencyNotFound();
        } catch (XPathExpressionException e) {
            breadbrumbDependencyNotFound();
        } catch (TransformerException e) {
            breadbrumbDependencyNotFound();
        }
    }

    /**
     * Modifica el layout por defecto del modulo web de la aplicacion para
     * permitirle hacer uso del breadcrumb.
     */
    private void editDefaultLayout() {

        String layoutPath =
            WebModuleUtils.getWebModuleFile(
                "src/main/webapp/WEB-INF/layouts/default.jspx",
                projectOperations).getAbsolutePath();
        FileUtilities.replaceText(XMLNS_UTIL, XMLNS_UTIL
            + " xmlns:breadcrumbs=\"" + getBreadcrumbsUrl() + "\"", layoutPath);
        FileUtilities
            .replaceText(
                "<div id=\"main\">",
                "<div id=\"main\">\n\t\t\t\t<breadcrumbs:breadcrumbs
parentPath=\"${parentPath}\"/>",
                layoutPath);
    }

    /**
     * Introduce en el fichero applicationContext-platform.xml los beans
     * configurados por el usuario para el uso de las migas de pan.
     */
    private void addPlatformBeans() {
```

```
        String newBeans =
            FileUtilities.readFileAsString(FileUtilities
                .getConfigurationFilePath("breadcrumbs-beans.xml"));
        newBeans =
            newBeans.replace(
                "</bean>",
                "<property name=\"scanPackage\" value=\""
                    + projectOperations.getFocusedTopLevelPackage()
                    + "\" />\n\t\t</bean>");
        String appContextPath =
            WebModuleUtils.getWebModuleFile(APP_CONTEXT_PLATFORM,
                projectOperations).getAbsolutePath();
        FileUtilities.replaceText("</beans>", "\t" + newBeans + "\n</beans>",
            appContextPath);
    }

    /**
     * Obtiene la URL a incluir en la cabecera del layout por defecto del modulo
     * web, para el uso de breadcrumb, a partir del fichero de configuracion
     * correspondiente.
     *
     * @return URL a incluir en la cabecera del layout por defecto del modulo
     *         web, para el uso de breadcrumb
     */
    private String getBreadcrumbsUrl() {

        BufferedReader br = null;
        String url = "";
        try {
            br =
                new BufferedReader(new FileReader(
                    FileUtilities
                        .getConfigurationFilePath("breadcrumbs-url.txt")));
            url = br.readLine();
        } catch (IOException ex) {
            LOG.log(Level.SEVERE,
                "ERROR. Configuration file \"breadcrumbs-url.txt\" not found");
        } finally {
```

```
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
        return url;
    }

    /**
     * Informa al usuario de que no se encontro el fichero de configuracion del
     * breadcrumb.
     */
    private void breadbrumbDependencyNotFound() {

        LOG.log(Level.SEVERE,
            "ERROR. File \"breadcrumb-dependency.xml\" not found in
configuration folder.");
    }
}
```

package-info.java

```
/**
 * Paquete para la configuracion de las "breadcrumb" o "migas de pan" de la
 * interfaz web de una aplicacion Spring Roo.
 */
package rooplusplus.roo.addon.breadcrumb;
```

Paquete Context

ContextCommands.java

```
package rooplusplus.roo.addon.context;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.osgi.service.component.ComponentContext;
import org.springframework.roo.shell.CliAvailabilityIndicator;
import org.springframework.roo.shell.CliCommand;
import org.springframework.roo.shell.CommandMarker;

/**
 * Comandos relacionados con el establecimiento de contextos (separacion
 * logica de los datos de la aplicacion, a los cuales cada usuario podra
 * acceder segun sus permisos). The command class is registered by the Roo
 * shell following an automatic classpath scan. You can provide simple user
 * presentation-related logic in this class. You can return any objects from
 * each method, or use the logger directly if you'd like to emit messages of
 * different severity (and therefore different colors on non-Windows systems).
 *
 * @since 1.1.1
 */
@Component
// Use these Apache Felix annotations to register your commands class in the Roo
// container
@Service
public class ContextCommands
    implements CommandMarker { // All command types must implement the

    // CommandMarker interface

    /**
     * Constructor de la clase.
```

```
     */
    public ContextCommands() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Get hold of a JDK Logger.
     */
    private static final Logger LOG = Logger.getLogger("ContextCommands.class");

    /**
     * Get a reference to the AddonOperations from the underlying OSGi
     * container.
     */
    @Reference
    private ContextOperations operations;

    /**
     * The activate method for this OSGi component, this will be called by the
     * OSGi container upon bundle activation (result of the 'addon install'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void activate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

    /**
     * The deactivate method for this OSGi component, this will be called by the
     * OSGi container upon bundle deactivation (result of the 'addon remove'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
```

```
    protected void deactivate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

    /**
     * Indica si es posible ejecutar el comando context setup en el proyecto que
     * tiene el foco.
     *
     * @return true si es posible ejecutar el comando en este momento; false en
     *         caso contrario
     */
    @CliAvailabilityIndicator("roo++ context setup")
    public boolean isContextSetupAvailable() {

        return operations.canContextSetup();
    }

    /**
     * Crea contextos (separacion logica de los datos de la aplicacion, a los
     * cuales cada usuario podra acceder segun sus permisos).
     */
    @CliCommand(value = "roo++ context setup", help = "Configures context for
user management")
    public void contextSetup() {

        operations.contextSetup();
        LOG.log(Level.INFO, "User management context configured");
    }
}
```

ContextOperations.java

```
package rooplusplus.roo.addon.context;

/**
 * Interface of commands that are available via the Roo shell.
 *
 * @since 1.1.1
 */
public interface ContextOperations {
```

```
    /**
     * Indica si es posible ejecutar el comando context setup en el proyecto que
     * tiene el foco.
     *
     * @return true si es posible ejecutar el comando en este momento; false en
     *         caso contrario
     */
    boolean canContextSetup();

    /**
     * Crea contextos (separacion logica de los datos de la aplicacion, a los
     * cuales cada usuario podra acceder segun sus permisos).
     */
    void contextSetup();
}
```

ContextOperationsImpl.java

```
package rooplusplus.roo.addon.context;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FilenameFilter;
import java.io.IOException;
import java.util.Locale;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.springframework.roo.project.ProjectOperations;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
import org.w3c.dom.Node;
import org.xml.sax.SAXException;

import rooplusplus.roo.addon.utils.FileUtilities;
import rooplusplus.roo.addon.utils.PomUtils;
import rooplusplus.roo.addon.utils.ProjectUtils;
import rooplusplus.roo.addon.utils.WebModuleUtils;
import rooplusplus.roo.addon.utils.XmlUtils;


/**
 * Implementation of {@link ContextOperations} interface.
 *
 * @since 1.1.1
 */
@Component
@Service
public class ContextOperationsImpl
    implements ContextOperations {

    /**
     * Constructor de la clase.
     */
    public ContextOperationsImpl() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Log para mostrar mensajes al usuario.
     */
    private static final Logger LOG = Logger
        .getLogger("ContextOperationsImpl.class");

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String DIRECTIVE_PAGE =
        "<jsp:directive.page
import=\"org.springframework.web.servlet.support.RequestContextUtils\" />";
```

```
    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String APP_CONTEXT_PORTAL =
        "src/main/resources/META-INF/spring/applicationContext-portal.xml";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String APP_CONTEXT_SECURITY =
        "src/main/resources/META-INF/spring/applicationContext-security.xml";

    /**
     * Get a reference to the ProjectOperations from the underlying OSGi
     * container. Make sure you are referencing the Roo bundle which contains
     * this service in your add-on pom.xml.
     */
    @Reference
    private ProjectOperations projectOperations;

    @Override
    public boolean canContextSetup() {

        return ProjectUtils.areModulesCreated(projectOperations)
            && WebModuleUtils.getWebModuleFile(
                "src/main/resources/META-INF/spring/application.properties",
                projectOperations).exists()
            && WebModuleUtils
                .getWebModuleFile(
                    "src/main/resources/META-INF/spring/applicationContext-
platform.xml",
                    projectOperations).exists()
            && WebModuleUtils.getWebModuleFile(APP_CONTEXT_PORTAL,
                projectOperations).exists()
            && WebModuleUtils.getWebModuleFile(APP_CONTEXT_SECURITY,
                projectOperations).exists()
            && WebModuleUtils.isWebJpa(projectOperations)
            && !WebModuleUtils.getWebModuleFile(
                "src/main/webapp/WEB-INF/views/contexts", projectOperations)
```

```
            .exists();
    }

    @Override
    public void contextSetup() {

        // Introducir en el pom.xml del modulo web las dependencias relacionadas
        // con los contextos
        addWebPomDependencies();
        // Modificar los ficheros applicationContext portal y security
        editApplicationContexts();
        // Introducir nuevas propiedades en portal.properties
        addNewPortalProperties();
        // Introducir nuevos mensajes de localizacion en messages.properties
        addNewMessagesProperties();
        // Actualizar el archivo header.jspx para el uso de los contextos
        updateHeader();
        // Copiar las vistas de los contextos
        FileUtilities.copyDirContent(
            FileUtilities.getConfigurationFilePath("contextsViews"),
            projectOperations.getFocusedProjectName()
                + "-web/src/main/webapp/WEB-INF/views/contexts");
        // Copiar el script SQL que es necesario ejecutar para poder hacer uso
        // de los contextos
        ProjectUtils.copyExternTextFile(
            ProjectUtils.getProjectPath(projectOperations),
            projectOperations.getFocusedProjectName(),
            "scripts/src/main/resources/context-dataload.sql",
            FileUtilities.getConfigurationFilePath("context-dataload.sql"));
    }

    /**
     * Introduce en el archivo pom.xml del modulo web las dependencias
     * relacionadas con el uso de contextos, segun lo configurado en
     * "context-dependencies.txt".
     */
    private void addWebPomDependencies() {

        try {
```

```
            doAddWebPomDependencies();
        } catch (IOException ex) {
            webPomDependenciesError();
        } catch (XPathExpressionException e) {
            webPomDependenciesError();
        } catch (ParserConfigurationException e) {
            webPomDependenciesError();
        } catch (SAXException e) {
            webPomDependenciesError();
        } catch (TransformerException e) {
            webPomDependenciesError();
        }
    }

    /**
     * Introduce en el archivo pom.xml del modulo web las dependencias
     * relacionadas con el uso de contextos, segun lo configurado en
     * "context-dependencies.txt". Si se producen excepciones, se las transmite
     * a addWebPomDependencies().
     *
     * @throws XPathExpressionException Si el contenido del pom.xml del modulo
     *         web no es el adecuado
     * @throws IOException Si context-dependencies.txt no existe o no tiene el
     *         contenido esperado
     * @throws ParserConfigurationException Si el contenido del pom.xml del
     *         modulo web no es el adecuado
     * @throws SAXException Si el contenido del pom.xml del modulo web no es el
     *         adecuado
     * @throws TransformerException Si el contenido del pom.xml del modulo web
     *         no es el adecuado
     */
    private void doAddWebPomDependencies()
        throws XPathExpressionException,
            IOException,
            ParserConfigurationException,
            SAXException,
            TransformerException {

        BufferedReader confFile = null;
```

```
        try {
            String pomPath =
                WebModuleUtils.getWebModuleFile("pom.xml", projectOperations)
                    .getAbsolutePath();
            confFile =
                new BufferedReader(new FileReader(
                    FileUtilities
                        .getConfigurationFilePath("context-dependencies.txt")));
            while (confFile.ready()) {
                String groupId = confFile.readLine();
                String artifactId = confFile.readLine();
                String version = confFile.readLine();
                if (groupId == null || artifactId == null || version == null) {
                    throw new IOException();
                }
                PomUtils.addDependency(pomPath, groupId, artifactId, version);
            }
        } finally {
            try {
                if (confFile != null) {
                    confFile.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Modifica los archivos applicationContext-portal.xml y
     * applicationContext-security.xml para el uso de los contextos.
     */
    private void editApplicationContexts() {

        editApplicationContextPortal();
        editApplicationContextSecurity();
    }

    /**
```

```
     * Introduce en el archivo applicationContext-portal.xml de un proyecto una
     * serie de beans, y una serie de nodos en sus lugares correspondientes
     * dentro del fichero, todo ello segun lo que indiquen los ficheros de
     * configuracion.
     */
    private void editApplicationContextPortal() {

        try {
            String appContextPath =
                WebModuleUtils.getWebModuleFile(APP_CONTEXT_PORTAL,
                    projectOperations).getAbsolutePath();
            addAllBeansToApplicationContext(appContextPath,
                FileUtilities.getConfigurationFilePath("contextBeans/portal"));
            Document doc = XmlUtils.createDocumentFromFile(appContextPath);
            File dir =
                new File(
                    FileUtilities

.getConfigurationFilePath("contextBeans/portal/nodesToAdd"));
            File[] list = dir.listFiles();
            if (list != null) {
                for (int i = 0; i < list.length; i++) {
                    doc = introduceNodo(doc, list[i]);
                }
            }
            XmlUtils.guardaXml(doc, appContextPath);
        } catch (IOException ex) {
            applicationContextPortalError();
        } catch (ParserConfigurationException e) {
            applicationContextPortalError();
        } catch (SAXException e) {
            applicationContextPortalError();
        } catch (XPathExpressionException e) {
            applicationContextPortalError();
        } catch (TransformerException e) {
            applicationContextPortalError();
        } catch (NullPointerException e) {
            applicationContextPortalError();
        }
    }
```

```
/**
 * Introduce un nodo es un documento XML.
 *
 * @param doc Documento XML en que se introducira el nodo
 * @param node Archivo que contiene los atributos del nodo: XPath de su
 *        padre, nombre, y pares atributo-valor, siempre cada cosa en una
 *        linea diferente
 * @return El documento con el nodo ya introducido
 * @throws XPathExpressionException Si el XPath no es correcto
 * @throws IOException Si el archivo con los datos del nodo no tiene el
 *         contenido esperado
 * @throws NullPointerException Si el XPath no es correcto
 */
private Document introduceNodo(Document doc, File node)
    throws XPathExpressionException,
        IOException, NullPointerException {

    BufferedReader br = null;
    try {
        br = new BufferedReader(new FileReader(node));
        String parentXPath = br.readLine();
        String nodeName = br.readLine();
        Node parent = XmlUtils.evaluateXPath(parentXPath, doc).item(0);
        Element newElement = doc.createElement(nodeName);
        parent.appendChild(newElement);
        while (br.ready()) {
            String attr = br.readLine();
            String value = br.readLine();
            if (attr == null || value == null) {
                throw new IOException();
            }
            newElement.setAttribute(attr, value);
        }
        return doc;
    } finally {
        try {
            if (br != null) {
                br.close();
```

```
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Modifica el archivo applicationContext-security.xml para el uso de los
     * contextos, introduciendo los beans necesarios, cambiando el
     * authenticationSuccessHandler y quitando la logout-success-url, ya que
     * esta URL han de proporcionarla los beans.
     */
    private void editApplicationContextSecurity() {

        String appContextPath =
            WebModuleUtils.getWebModuleFile(APP_CONTEXT_SECURITY,
                projectOperations).getAbsolutePath();
        XmlUtils
            .eliminaNodoSiExiste(appContextPath, "//*[name()='sec:logout']");
        FileUtilities
            .replaceText(
                "use-expressions=\"true\">",
                "use-expressions=\"true\">\n\n\t\t<sec:logout success-handler-
ref=\"contextLogoutSuccessHandler\" invalidate-session=\"true\" logout-
url=\"/j_spring_security_logout\" />",
                appContextPath);
        addAllBeansToApplicationContext(appContextPath,
            FileUtilities.getConfigurationFilePath("contextBeans/security"));
        try {
            FileUtilities
                .replaceText(
                    "<property name=\"authenticationSuccessHandler\"
ref=\"authenticationSuccessHandler\"/>",
                    "<property name=\"authenticationSuccessHandler\" ref=\""
                        + FileUtilities

.getFirstLineConfFile("contextBeans/security/authenticationSuccessHandler.txt")
                        + "\" />", appContextPath);
        } catch (IOException e) {
            LOG.log(
```

```
                Level.SEVERE,
                "ERROR. Configuration
file \"contextBeans/security/authenticationSuccessHandler.txt\" not found or
empty.");
        }
    }

    /**
     * Introduce un un archivo XML de tipo applicationContext todos los beans
     * contenidos en los distintos ficheros XML situados en un directorio.
     *
     * @param appContextPath Ruta del archivo XML de tipo applicationContext
     * @param beansDirPath Ruta del directorio con los beans en archivos XML
     */
    private void addAllBeansToApplicationContext(String appContextPath,
                                                 String beansDirPath) {

        File dir = new File(beansDirPath);
        File[] files = dir.listFiles(new FilenameFilter() {

            @Override
            public boolean accept(File dir, String name) {

                return name.toLowerCase(Locale.getDefault()).endsWith(".xml");
            }
        });
        for (int i = 0; i < files.length; i++) {
            String bean =
                FileUtilities.readFileAsString(files[i].getAbsolutePath());
            FileUtilities.replaceText("</beans>", bean + "\n</beans>",
                appContextPath);
        }
    }

    /**
     * Introduce en portal.properties las propiedades relacionadas con los
     * contextos.
     */
    private void addNewPortalProperties() {
```

```
        try {
            String newProperties =
                FileUtilities.readFileAsStringThrows(FileUtilities
                    .getConfigurationFilePath("context-portal.properties"));
            ProjectUtils.appendTextToFile(newProperties,
                ProjectUtils.getProjectPath(projectOperations),
                projectOperations.getFocusedProjectName(),
                "web/src/main/resources/META-INF/portal.properties");
        } catch (IOException ex) {
            LOG.log(Level.SEVERE,
                "ERROR. Configuration file \"context-portal.properties\" not
found.");
        }
    }

    /**
     * Introduce en messages.properties los mensajes de localizacion
     * relacionados con los contextos.
     */
    private void addNewMessagesProperties() {

        try {
            String newProperties =
                FileUtilities.readFileAsStringThrows(FileUtilities
                    .getConfigurationFilePath("context-messages.properties"));
            ProjectUtils.appendTextToFile(newProperties,
                ProjectUtils.getProjectPath(projectOperations),
                projectOperations.getFocusedProjectName(),
                "web/src/main/webapp/WEB-INF/i18n/messages.properties");
        } catch (IOException ex) {
            LOG.log(Level.SEVERE,
                "ERROR. Configuration file \"context-messages.properties\" not
found.");
        }
    }

    /**
     * Informa al usuario de que se ha producido un error al intentar introducir
     * las dependencias en el pom.xml del modulo web.
     */
```

```
    private void webPomDependenciesError() {

        LOG.log(
            Level.SEVERE,
            "ERROR. Configuration file \"context-dependencies.txt\" not found or
with wrong content.");
    }

    /**
     * Informa de que los archivos de configuracion que indican como modificar
     * applicationContext-portal.xml no tienen un contenido correcto.
     */
    private void applicationContextPortalError() {

        LOG.log(
            Level.SEVERE,
            "ERROR. Files in \"contextBeans/portal/nodesToAdd\" have wrong
content (possibly XPath is not valid).");
    }

    /**
     * Actualiza el archivo header.jspx para incluir el uso de los contextos.
     */
    private void updateHeader() {

        try {
            String headerPath =
                WebModuleUtils.getWebModuleFile(
                    "src/main/webapp/WEB-INF/views/header.jspx",
                    projectOperations).getAbsolutePath();
            String newDirectivePages =
                FileUtilities
                    .readFileAsStringThrows(FileUtilities
                        .getConfigurationFilePath("contextHeader/directive-
pages.jspx"));
            String newScriptlets =
                FileUtilities.readFileAsStringThrows(FileUtilities
                    .getConfigurationFilePath("contextHeader/scriptlets.jspx"));
            String addToPortalHeader =
                FileUtilities
```

```
                    .readFileAsStringThrows(FileUtilities
                        .getConfigurationFilePath("contextHeader/add-to-portal-
header.txt"));
            FileUtilities.replaceText(DIRECTIVE_PAGE, DIRECTIVE_PAGE + "\n"
                + newDirectivePages, headerPath);
            FileUtilities.replaceText("</jsp:scriptlet>", "</jsp:scriptlet>\n"
                + newScriptlets, headerPath);
            FileUtilities
                .replaceText("requestUri=\"${requestUri}\"",
                    "requestUri=\"${requestUri}\" " + addToPortalHeader,
                    headerPath);
        } catch (IOException ex) {
            LOG.log(
                Level.SEVERE,
                "ERROR. Configuration file \"contextHeader/directive-
pages.jspx\", \"\" orcontextHeader/scriptlets.jspx or \"add-to-portal-
header.txt\" not found.");
        }
    }
}
```

package-info.java

```
/**
 * Package Javadoc.
 */
package rooplusplus.roo.addon.context;
```

Paquete Layout

LayoutCommands.java

```
package rooplusplus.roo.addon.layout;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.osgi.service.component.ComponentContext;
import org.springframework.roo.shell.CliAvailabilityIndicator;
import org.springframework.roo.shell.CliCommand;
import org.springframework.roo.shell.CommandMarker;

/**
 * Comandos relacionados con el layout de la aplicacion del modulo web de los
 * proyectos. The command class is registered by the Roo shell following an
 * automatic classpath scan. You can provide simple user presentation-related
 * logic in this class. You can return any objects from each method, or use the
 * logger directly if you'd like to emit messages of different severity (and
 * therefore different colors on non-Windows systems).
 *
 * @since 1.1.1
 */
@Component
// Use these Apache Felix annotations to register your commands class in the Roo
// container
@Service
public class LayoutCommands
    implements CommandMarker { // All command types must implement the

    // CommandMarker interface

    /**
     * Constructor de la clase.
     */
```

```
public LayoutCommands() {

    // Todas las clases deben tener constructor.
}

/**
 * Get hold of a JDK Logger.
 */
private static final Logger LOG = Logger.getLogger("LayoutCommands.class");

/**
 * Get a reference to the AddonOperations from the underlying OSGi
 * container.
 */
@Reference
private LayoutOperations operations;

/**
 * The activate method for this OSGi component, this will be called by the
 * OSGi container upon bundle activation (result of the 'addon install'
 * command).
 *
 * @param context the component context can be used to get access to the
 *        OSGi container (ie find out if certain bundles are active)
 */
protected void activate(ComponentContext context) {

    // Es necesario que exista este metodo
}

/**
 * The deactivate method for this OSGi component, this will be called by the
 * OSGi container upon bundle deactivation (result of the 'addon remove'
 * command).
 *
 * @param context the component context can be used to get access to the
 *        OSGi container (ie find out if certain bundles are active)
 */
protected void deactivate(ComponentContext context) {
```

```
        // Es necesario que exista este metodo
    }

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

        return operations.canLayoutSetup();
    }

    /**
     * Configura el layout (apariencia, estilo) del modulo web del proyecto que
     * tiene el foco.
     */
    @CliCommand(value = "roo++ layout setup", help = "Configures web app
layout")
    public void layoutSetup() {

        operations.layoutSetup();
        LOG.log(Level.INFO, "Web app layout configured");
    }
}
```

LayoutOperations.java

```
package rooplusplus.roo.addon.layout;

/**
 * Interface of commands that are available via the Roo shell.
 *
 * @since 1.1.1
 */
public interface LayoutOperations {
```

```
    /**
     * Indica si es posible ejecutar el comando layout setup en el proyecto que
     * tiene el foco.
     *
     * @return true si es posible ejecutar el comando en este momento; false en
     *         caso contrario
     */
    boolean canLayoutSetup();

    /**
     * Configura el layout (apariencia, estilo) del modulo web del proyecto que
     * tiene el foco.
     */
    void layoutSetup();
}
```

LayoutOperationsImpl.java

```
package rooplusplus.roo.addon.layout;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.springframework.roo.project.ProjectOperations;

import rooplusplus.roo.addon.utils.FileUtilities;
import rooplusplus.roo.addon.utils.ProjectUtils;
import rooplusplus.roo.addon.utils.WebModuleUtils;



/**
 * Implementation of {@link LayoutOperations} interface.
 *
 * @since 1.1.1
 */
@Component
@Service
```

```
public class LayoutOperationsImpl
    implements LayoutOperations {

    /**
     * Constructor de la clase.
     */
    public LayoutOperationsImpl() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Log para mostrar mensajes al usuario.
     */
    private static final Logger LOG = Logger
        .getLogger("LayoutOperationsImpl.class");

    /**
     * Cadena con el caracter "/", usado en varias ocasiones en la clase.
     */
    private static final String BARRA = "/";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String IMAGES = "images";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String STYLES = "styles";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String SCRIPTS = "scripts";

    /**
     * Cadena usada en varias ocasiones en la clase.
```

```
     */
    private static final String WEBMVC_CONFIG =
        "src/main/webapp/WEB-INF/spring/webmvc-config.xml";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String APPLICATION_PROPERTIES =
        "src/main/resources/META-INF/spring/application.properties";

    /**
     * Get a reference to the ProjectOperations from the underlying OSGi
     * container. Make sure you are referencing the Roo bundle which contains
     * this service in your add-on pom.xml.
     */
    @Reference
    private ProjectOperations projectOperations;

    @Override
    public boolean canLayoutSetup() {

        return ProjectUtils.areModulesCreated(projectOperations)
            && WebModuleUtils.getWebModuleFile(APPLICATION_PROPERTIES,
                projectOperations).exists()
            && WebModuleUtils
                .getWebModuleFile(WEBMVC_CONFIG, projectOperations).exists();
    }

    @Override
    public void layoutSetup() {

        // Obtener directorios de configuracion y webapp
        String confDir = FileUtilities.getConfigurationFilePath("layout");
        String webappDir =
            WebModuleUtils.getWebModuleFile("src/main/webapp",
                projectOperations).getAbsolutePath();
        // Copiar los archivos necesarios para el layout
        copyLayoutFiles(confDir, webappDir);
        // Modificar load-scripts.jspx para cargar correctamente CSS y
```

```
        // JavaScript
        modifyLoadScripts(confDir, webappDir);
        // Incluir variable application.theme en application.properties
        String appProperties = APPLICATION_PROPERTIES;
        if (!FileUtilities.readFileAsString(
            WebModuleUtils.getWebModuleFile(appProperties, projectOperations)
                .getAbsolutePath()).contains("application.theme")) {
            ProjectUtils.appendTextToFile("application.theme=standard",
                ProjectUtils.getProjectPath(projectOperations),
                projectOperations.getFocusedProjectName(), "web/"
                    + appProperties);
        }
        // Paquete el bean CookieThemeResolver por FixedThemeResolver
        replaceThemeResolver();
        // Eliminar <util:theme/> en footer.jspx
        removeUtilTheme();
    }

    /**
     * Copiar los archivos necesarios para el layout.
     *
     * @param confDir Ruta del directorio de configuracion del layout
     * @param webappDir Ruta del directorio webapp del modulo web del proyecto
     */
    private void copyLayoutFiles(String confDir, String webappDir) {

        String[] dirs =
            {IMAGES, STYLES, SCRIPTS, "layouts", "tags", "properties",
                "classes"};
        String[] destDirs =
            {IMAGES, STYLES, SCRIPTS, "WEB-INF/layouts", "WEB-INF/tags",
                "WEB-INF/i18n", "WEB-INF/classes"};
        for (int i = 0; i < destDirs.length; i++) {
            destDirs[i] = webappDir + BARRA + destDirs[i];
        }
        for (int i = 0; i < dirs.length; i++) {
            FileUtilities
                .copyDirContent(confDir + BARRA + dirs[i], destDirs[i]);
        }
```

```
    }

    /**
     * Modifica el fichero load-scripts.jspx del modulo web del proyecto para
     * que carguen correctamente CSS y JavaScript.
     *
     * @param confDir Ruta del directorio de configuracion del layout
     * @param webappDir Ruta del directorio webapp del modulo web del proyecto
     */
    private void modifyLoadScripts(String confDir, String webappDir) {

        String loadScripts = webappDir + "/WEB-INF/tags/util/load-scripts.tagx";
        String newText =
            FileUtilities.readFileAsString(confDir + "/load-scripts.jspx");
        if ("".equals(newText)) {
            LOG.log(
                Level.WARNING,
                "WARINING. \"load-scripts.jspx\" file not found in configuration
directory or empty");
        } else {
            // Si se habia hecho layout setup con el mismo load-scripts.jspx
            // avisamos al usuario y salimos
            if (FileUtilities.readFileAsString(loadScripts).contains(newText)) {
                LOG.log(
                    Level.WARNING,
                    "WARNING. Layout setup had been executed previously with the
same \"load-scripts.jspx\" file.");
                return;
            }
            // Introducir el nuevo texto en load-scripts.jspx
            FileUtilities.replaceText("</jsp:root>", newText + "\n</jsp:root>",
                loadScripts);
        }
        String loadScriptsText = FileUtilities.readFileAsString(loadScripts);
        String texto =
            "<!--If portal is configured, load-scripts must go here. DO NOT
MANUALLY EDIT OR REMOVE THIS LINE if you want to run portal setup in the
future-->\n";
        if (!loadScriptsText
            .contains("<portal:load-scripts noIncludeDojo=\"true\"/>")
            && !loadScriptsText.contains(texto)) {
```

```
            FileUtilities.replaceText(newText, texto + newText, loadScripts);
        }
    }

    /**
     * Reemplaza el bean CookieThemeResolver por FixedThemeResolver.
     */
    private void replaceThemeResolver() {

        String webMvcPath =
            WebModuleUtils.getWebModuleFile(WEBMVC_CONFIG, projectOperations)
                .getAbsolutePath();
        FileUtilities.replaceText(
            "<!-- Store preferred theme configuration in a cookie -->",
            "<!-- Fixed theme resolver -->", webMvcPath);
        FileUtilities
            .replaceText(
                "<bean
class=\"org.springframework.web.servlet.theme.CookieThemeResolver\"
id=\"themeResolver\" p:cookieName=\"theme\" p:defaultThemeName=\"standard\"/>",
                "<bean
class=\"org.springframework.web.servlet.theme.FixedThemeResolver\"
id=\"themeResolver\" p:defaultThemeName=\"${application.theme}\"/>",
                webMvcPath);
    }

    /**
     * Elimina el nodo <util:theme/> en el archivo footer.jspx, en el modulo web
     * del proyecto que tiene el foco.
     */
    private void removeUtilTheme() {

        FileUtilities.replaceText(
            "<util:theme/>",
            "",
            WebModuleUtils.getWebModuleFile(
                "src/main/webapp/WEB-INF/views/footer.jspx", projectOperations)
                .getAbsolutePath());
    }
}
```

package-info.java

```
/**
 * Paquete para la configuracion de la apariencia de la interfaz web de una
 * aplicacion Spring Roo.
 */
package rooplusplus.roo.addon.layout;
```

Paquete Platform

PlatformCommands.java

```
package rooplusplus.roo.addon.platform;

import java.io.File;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.osgi.service.component.ComponentContext;
import org.springframework.roo.shell.CliAvailabilityIndicator;
import org.springframework.roo.shell.CliCommand;
import org.springframework.roo.shell.CommandMarker;

import rooplusplus.roo.addon.utils.FileUtilities;


/**
 * Comandos relacionados con el uso de una plataforma externa en los proyectos.
 * The command class is registered by the Roo shell following an automatic
 * classpath scan. You can provide simple user presentation-related logic in
 * this class. You can return any objects from each method, or use the logger
 * directly if you'd like to emit messages of different severity (and therefore
 * different colors on non-Windows systems).
 *
 * @since 1.1.1
 */
@Component
// Use these Apache Felix annotations to register your commands class in the Roo
```

```
// container
@Service
public class PlatformCommands
    implements CommandMarker { // All command types must implement the

    // CommandMarker interface

    /**
     * Constructor de la clase.
     */
    public PlatformCommands() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Get hold of a JDK Logger.
     */
    private static final Logger LOG = Logger
        .getLogger("PlatformCommands.class");

    /**
     * Ruta interna, dentro del directorio "resources" del JAR, del directorio
     * donde se encuentran realmente todos los resources.
     */
    private static final String INTERNALPATH = "rooAddon/";

    /**
     * Get a reference to the AddonOperations from the underlying OSGi
     * container.
     */
    @Reference
    private PlatformOperations operations;

    /**
     * The activate method for this OSGi component, this will be called by the
     * OSGi container upon bundle activation (result of the 'addon install'
     * command).
     *
```

```
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void activate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

    /**
     * The deactivate method for this OSGi component, this will be called by the
     * OSGi container upon bundle deactivation (result of the 'addon remove'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void deactivate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

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

        return operations.canPlatformSetup();
    }

    /**
     * Introduce en el proyecto que tiene el foco dependencias respecto a una
     * plataforma configurada por el usuario.
     */
    @CliCommand(value = "roo++ platform setup", help = "Adds platform
dependencies to project")
```

```
    public void platformSetup() {

        operations.platformSetup();
        LOG.log(Level.INFO, "Platform dependencies added to project");
    }

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

        return (new File(
            FileUtilities
                .getConfigurationFilePath("applicationContext-platform.xml")))
            .exists()
            && operations.isPlatformSetup()
            && !operations.isPlatformConfigured();
    }

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

        operations.platformConfiguration(INTERNALPATH);
        LOG.log(Level.INFO,
            "Platform configuration beans added to project's web module");
    }
}
```

PlatformOperations.java

```
package rooplusplus.roo.addon.platform;
```

```
/**
 * Interface of commands that are available via the Roo shell.
 *
 * @since 1.1.1
 */
public interface PlatformOperations {

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

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

    /**
     * Determina si ya se han introducido en el proyecto que tiene el foco
     * dependencias con respecto a una plataforma.
     *
     * @return true si ya se han introducido; false en caso contrario
     */
    boolean isPlatformSetup();

    /**
     * Determina si ya se han introducido en el modulo web del proyecto que
     * tiene el foco los beans necesarios para el uso de la plataforma
     * anteriormente configurada.
     *
     * @return true si ya se han introducido; false en caso contrario
     */
    boolean isPlatformConfigured();
```

```
    /**
     * Introduce en el modulo web del proyecto que tiene el foco los beans
     * necesarios para el uso de la plataforma anteriormente configurada.
     *
     * @param internalPath Ruta interna, dentro del directorio "resources" del
     *        JAR, del directorio donde se encuentran realmente todos los
     *        resources.
     */
    void platformConfiguration(String internalPath);
}
```

PlatformOperationsImpl.java

```
package rooplusplus.roo.addon.platform;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.springframework.roo.process.manager.FileManager;
import org.springframework.roo.project.ProjectOperations;
import org.xml.sax.SAXException;

import rooplusplus.roo.addon.utils.FileUtilities;
import rooplusplus.roo.addon.utils.PomUtils;
import rooplusplus.roo.addon.utils.ProjectUtils;
import rooplusplus.roo.addon.utils.WebModuleUtils;
```

```
/**
 * Implementation of {@link PlatformOperations} interface.
 *
 * @since 1.1.1
 */
@Component
@Service
public class PlatformOperationsImpl
    implements PlatformOperations {

    /**
     * Constructor de la clase.
     */
    public PlatformOperationsImpl() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Log para mostrar informacion al usuario.
     */
    private static final Logger LOG = Logger
        .getLogger("PlatformOperationsImpl.class");

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String COMA = ",";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String PLATFORM_CONFFILE = "platform.txt";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String POM = "/pom.xml";
```

```
    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String BASE_PACKAGE =
        "<context:component-scan base-package=\"";

    /**
     * Get a reference to the FileManager from the underlying OSGi container.
     * Make sure you are referencing the Roo bundle which contains this service
     * in your add-on pom.xml. Using the Roo file manager instead if
     * java.io.File gives you automatic rollback in case an Exception is thrown.
     */
    @Reference
    private FileManager fileManager;

    /**
     * Get a reference to the ProjectOperations from the underlying OSGi
     * container. Make sure you are referencing the Roo bundle which contains
     * this service in your add-on pom.xml.
     */
    @Reference
    private ProjectOperations projectOperations;

    @Override
    public boolean canPlatformSetup() {

        return (new File(
            FileUtilities.getConfigurationFilePath(PLATFORM_CONFFILE)).exists()
            && ProjectUtils.areModulesCreated(projectOperations) && !
isPlatformSetup());
    }

    // Incluir en los pom.xml de los modulos raiz, core y web dependencias con
    // una plataforma que contiene utilidades para ser utilizadas dichos modulos
    @Override
    public void platformSetup() {

        try {
            // Leer parametros del fichero de configuracion asociado
            String[] confFileData = readConfFile();
```

```
            String groupId = confFileData[0], version = confFileData[1],
testModule =
                confFileData[2];
            String[] groupIdSplit = groupId.split("\\.");
            String platformName = groupIdSplit[groupIdSplit.length - 1];
            // Introducir propiedad en pom raiz y depenencias en poms core y web
            addPlatformPropertyAndDepenencies(version, groupId, testModule,
                platformName);
        } catch (IOException ex) {
            error();
        } catch (XPathExpressionException e) {
            error();
        } catch (ParserConfigurationException e) {
            error();
        } catch (SAXException e) {
            error();
        } catch (TransformerException e) {
            error();
        }
    }

    /**
     * Introduce la propiedad "platform.version" en el pom.xml raiz, y las
     * dependencias respecto a la plataforma en los pom.xml de los modulos core
     * y web.
     *
     * @param version Version de la plataforma a utilizar
     * @param groupId Campo groupId de la plataforma
     * @param testModule Nombre del modulo de test de la plataforma
     * @param platformName Nombre de la plataforma
     * @throws XPathExpressionException Si el pom.xml no existe en el lugar
     *         esperado o no tiene el contenido adecuado
     * @throws ParserConfigurationException Si el pom.xml no existe en el lugar
     *         esperado o no tiene el contenido adecuado
     * @throws SAXException Si el pom.xml no existe en el lugar esperado o no
     *         tiene el contenido adecuado
     * @throws IOException Si el pom.xml no existe en el lugar esperado o no
     *         tiene el contenido adecuado
     * @throws TransformerException Si el pom.xml no existe en el lugar esperado
     *         o no tiene el contenido adecuado
```

```
     */
    private void addPlatformPropertyAndDepenencies(String version,
                                                   String groupId,
                                                   String testModule,
                                                   String platformName)
        throws XPathExpressionException,
            ParserConfigurationException,
            SAXException,
            IOException,
            TransformerException {

        // Introducir en el pom raiz la version de la plataforma a utilizar
        String pomPath = ProjectUtils.getProjectPath(projectOperations) + POM;
        PomUtils.addProperty(pomPath, "platform.version", version);
        // Introducir en pom core y web dependencias respecto a la plataforma
        PomUtils.addPlatformDependency("core", projectOperations, fileManager,
            platformName, groupId, testModule);
        PomUtils.addPlatformDependency("web", projectOperations, fileManager,
            platformName, groupId, testModule);
    }

    /**
     * Lee los atributos de la plataforma desde el fichero de configuracion
     * correspondiente.
     *
     * @return Campos groupId, version y modulo de test de la plataforma,
     *         respectivamente
     * @throws IOException Si no fue posible leer alguno de los 3 campos del
     *         fichero de configuracion
     */
    private String[] readConfFile() throws IOException {

        BufferedReader confFile = null;
        try {
            String confFilePath =
                FileUtilities.getConfigurationFilePath(PLATFORM_CONFFILE);
            confFile = new BufferedReader(new FileReader(confFilePath));
            String groupId = confFile.readLine();
            String version = confFile.readLine();
```

```
            String testModule = confFile.readLine();
            confFile.close();
            if (groupId == null || version == null || testModule == null) {
                throw new IOException();
            }
            return new String[] {groupId, version, testModule};
        } finally {
            if (confFile != null) {
                confFile.close();
            }
        }
    }

    @Override
    public boolean isPlatformSetup() {

        String pomPath = ProjectUtils.getProjectPath(projectOperations) + POM;
        String pom = FileUtilities.readFileAsString(pomPath);
        return pom.contains("</platform.version>");
    }

    @Override
    public boolean isPlatformConfigured() {

        return WebModuleUtils
            .getWebModuleFile(
                "src/main/resources/META-INF/spring/applicationContext-
platform.xml",
                projectOperations).exists();
    }

    @Override
    public void platformConfiguration(String internalPath) {

        WebModuleUtils.copyApplicationContextPlatformToWebModule(
            ProjectUtils.getProjectPath(projectOperations),
            projectOperations.getFocusedProjectName());
        // Introducir como base-package en component-scan (dentro de
        // applicationContext.xml, y de webmvc-config.xml) la plataforma
        try {
```

```
            addPlatformBasePackage();
        } catch (IOException ex) {
            error();
        }
    }

    /**
     * Introduce el groupId de la plataforma como base-package en
     * component-scan, junto con el del propio proyecto, que ya figuraba, en los
     * ficheros applicationContext.xml y webmvc-config.xml.
     *
     * @throws IOException Si no existe o no tiene el contenido correcto alguno
     *         de los ficheros: platform.txt, applicationContext.xml,
     *         webmvc-config.xml
     */
    private void addPlatformBasePackage() throws IOException {

        String groupId = FileUtilities.getFirstLineConfFile(PLATFORM_CONFFILE);
        String topLevelPackage =
            projectOperations.getFocusedTopLevelPackage()
                .getFullyQualifiedPackageName();
        FileUtilities.replaceText(
            BASE_PACKAGE + topLevelPackage,
            BASE_PACKAGE + topLevelPackage + COMA + groupId,
            WebModuleUtils.getWebModuleFile(
                "src/main/resources/META-INF/spring/applicationContext.xml",
                projectOperations).getAbsolutePath());
        FileUtilities.replaceText(
            BASE_PACKAGE + topLevelPackage,
            BASE_PACKAGE + topLevelPackage + COMA + groupId,
            WebModuleUtils.getWebModuleFile(
                "src/main/webapp/WEB-INF/spring/webmvc-config.xml",
                projectOperations).getAbsolutePath());
    }

    /**
     * Informa al usuario de que no se ha encontrado el fichero de configuracion
     * requerido.
     */
```

```
    private void error() {

        LOG.log(
            Level.SEVERE,
            "ERROR. Required configuration file \"platform.txt\" not found, or
with wrong content");
    }
}
```

package-info.java

```
/**
 * Paquete para la configuracion del uso de una plataforma externa desde una
 * aplicacion Spring Roo.
 */
package rooplusplus.roo.addon.platform;
```

Paquete Portal

PortalCommands.java

```
package rooplusplus.roo.addon.portal;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.osgi.service.component.ComponentContext;
import org.springframework.roo.shell.CliAvailabilityIndicator;
import org.springframework.roo.shell.CliCommand;
import org.springframework.roo.shell.CommandMarker;

/**
 * Comandos relacionados con el uso de un portal en el modulo web de los
 * proyectos. The command class is registered by the Roo shell following an
 * automatic classpath scan. You can provide simple user presentation-related
 * logic in this class. You can return any objects from each method, or use the
 * logger directly if you'd like to emit messages of different severity (and
 * therefore different colors on non-Windows systems).
 *
```

```
 * @since 1.1.1
 */
@Component
// Use these Apache Felix annotations to register your commands class in the Roo
// container
@Service
public class PortalCommands
    implements CommandMarker { // All command types must implement the

    // CommandMarker interface

    /**
     * Constructor de la clase.
     */
    public PortalCommands() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Get hold of a JDK Logger.
     */
    private static final Logger LOG = Logger.getLogger("PortalCommands.class");

    /**
     * Ruta interna, dentro del directorio "resources" del JAR, del directorio
     * donde se encuentran realmente todos los resources.
     */
    private static final String INTERNALPATH = "rooAddon/";

    /**
     * Get a reference to the AddonOperations from the underlying OSGi
     * container.
     */
    @Reference
    private PortalOperations operations;

    /**
     * The activate method for this OSGi component, this will be called by the
```

```
     * OSGi container upon bundle activation (result of the 'addon install'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void activate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

    /**
     * The deactivate method for this OSGi component, this will be called by the
     * OSGi container upon bundle deactivation (result of the 'addon remove'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void deactivate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

    /**
     * Indica si es posible ejecutar el comando portal setup en el proyecto que
     * tiene el foco.
     *
     * @return true si es posible ejecutar el comando en este momento; false en
     *         caso contrario
     */
    @CliAvailabilityIndicator("roo++ portal setup")
    public boolean isPortalSetupAvailable() {

        return operations.isPortalSetupAvailable();
    }

    /**
     * Configura el modulo web del proyecto que tiene el foco para el uso del
```

```
     * portal de la organizacion, segun lo configurado por el usuario.
     */
    @CliCommand(value = "roo++ portal setup", help = "Configures web module to
use organization's portal")
    public void portalSetup() {

        operations.portalSetup(INTERNALPATH);
        LOG.log(Level.INFO, "Portal configured for web module");
    }
}
```

PortalOperations.java

```
package rooplusplus.roo.addon.portal;

/**
 * Interface of commands that are available via the Roo shell.
 *
 * @since 1.1.1
 */
public interface PortalOperations {

    /**
     * Configura el modulo web del proyecto que tiene el foco para el uso del
     * portal de la organizacion, segun lo configurado por el usuario.
     *
     * @param internalPath uta interna, dentro del directorio "resources" del
     *        JAR, del directorio donde se encuentran realmente todos los
     *        resources.
     */
    void portalSetup(String internalPath);

    /**
     * Indica si es posible ejecutar el comando portal setup en el proyecto que
     * tiene el foco.
     *
     * @return true si es posible ejecutar el comando en este momento; false en
     *         caso contrario
     */
    boolean isPortalSetupAvailable();
}
```

PortalOperationsImpl.java

```
package rooplusplus.roo.addon.portal;

/**
 * Interface of commands that are available via the Roo shell.
 *
 * @since 1.1.1
 */
public interface PortalOperations {

    /**
     * Configura el modulo web del proyecto que tiene el foco para el uso del
     * portal de la organizacion, segun lo configurado por el usuario.
     *
     * @param internalPath uta interna, dentro del directorio "resources" del
     *        JAR, del directorio donde se encuentran realmente todos los
     *        resources.
     */
    void portalSetup(String internalPath);

    /**
     * Indica si es posible ejecutar el comando portal setup en el proyecto que
     * tiene el foco.
     *
     * @return true si es posible ejecutar el comando en este momento; false en
     *         caso contrario
     */
    boolean isPortalSetupAvailable();
}
```

package-info.java

```
/**
 * Paquete para la configuracion de un portal para el modulo web de una
 * aplicacion Spring Roo.
 */
package rooplusplus.roo.addon.portal;
```

Paquete Project

ProjectCommands.java

```
package rooplusplus.roo.addon.project;

import java.io.File;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.osgi.service.component.ComponentContext;
import org.springframework.roo.project.ProjectOperations;
import org.springframework.roo.shell.CliAvailabilityIndicator;
import org.springframework.roo.shell.CliCommand;
import org.springframework.roo.shell.CliOption;
import org.springframework.roo.shell.CommandMarker;

import rooplusplus.roo.addon.utils.FileUtilities;


/**
 * Comandos relacionados con la creacion y configuracion de proyectos. The
 * command class is registered by the Roo shell following an automatic classpath
 * scan. You can provide simple user presentation-related logic in this class.
 * You can return any objects from each method, or use the logger directly if
 * you'd like to emit messages of different severity (and therefore different
 * colors on non-Windows systems).
 *
 * @since 1.1.1
 */
@Component
// Use these Apache Felix annotations to register your commands class in the Roo
// container
@Service
public class ProjectCommands
    implements CommandMarker { // All command types must implement the
```

```
// CommandMarker interface

/**
 * Constructor de la clase.
 */
public ProjectCommands() {

    // Todas las clases deben tener constructor.
}

/**
 * Get hold of a JDK Logger.
 */
private static final Logger LOG = Logger.getLogger("ProjectCommands.class");

/**
 * Get a reference to the AddonOperations from the underlying OSGi
 * container.
 */
@Reference
private ProjectOps operations;

/**
 * Permite conocer cuales son el proyecto y el modulo que tienen el foco.
 */
@Reference
private ProjectOperations projectOperations;

/**
 * Ruta interna, dentro del directorio "resources" del JAR, del directorio
 * donde se encuentran realmente todos los resources.
 */
private static final String INTERNALPATH = "rooAddon/";

/**
 * The activate method for this OSGi component, this will be called by the
 * OSGi container upon bundle activation (result of the 'addon install'
 * command).
```

```
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void activate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

    /**
     * The deactivate method for this OSGi component, this will be called by the
     * OSGi container upon bundle deactivation (result of the 'addon remove'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void deactivate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

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

        return projectOperations.isFocusedProjectAvailable()
            && !operations.areModulesCreated();
    }

    /**
     * Crea los diferentes modulos del proyecto Maven/Spring Roo que tiene el
     * foco.
```

```
     *
     * @param version Nombre de la version que el usuario asigna al proyecto
     *        para el que se estan creando los modulos
     */
    @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)
{

        operations.createModules(projectOperations.getFocusedProjectName(),
            version, INTERNALPATH);
        LOG.log(Level.INFO, "All project modules created");
    }

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

        // La licencia puede ser introducida si los modulos han sido
        // creados y no se habia introducido previamente.
        return operations.areModulesCreated() && !operations
                .isLicenseIntroduced();
    }

    /**
     * Configura la licencia del proyecto que tiene el foco.
     */
    @CliCommand(value = "roo++ project license", help = "Configures the license
for a roo++ project")
    public void license() {

        operations.license(INTERNALPATH);
        LOG.log(Level.INFO, "Project license configured");
    }
```

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

        return operations.areModulesCreated() && !operations.isWebConfigured();
    }

    /**
     * Realiza diferentes acciones de configuracion sobre el modulo web del
     * proyecto que tiene el foco.
     */
    @CliCommand(value = "roo++ project web configuration", help = "Configures
the web module for a roo++ project")
    public void webConfiguration() {

        operations.webConfiguration(INTERNALPATH);
        LOG.log(Level.INFO, "Web module configured");
    }

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

        return operations.areModulesCreated() && !operations.isWebJpa();
    }

    /**
```

```
     * Configura el modulo "web" del proyecto que tiene el foco para el uso de
     * una base de datos.
     *
     * @param user Nombre de usuario para el acceso a la base de datos
     * @param password Password para el acceso a la base de datos
     * @param database Nombre de la base de datos (que sera el nombre del
     *        proyecto si no se indica)
     * @param dbms Nombre del sistema de gestion de bases de datos: oracle
     *        (valor por defecto), mysql, mssql, postgres
     * @param server Maquina que contiene la base de datos
     * @param portNumber Puerto a traves del cual se accede a la base de datos
     * @param hibernateVersion Version de Hibernate a utilizar
     */
    @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) {


        String port = portNumber;
        // Si el puerto no es un numero entero pasa a ser la cadena vacia, que
        // corresponde al
        // puerto por defecto del SGBD utilizado
        if (!port.matches("-?\\d+")) {
            port = "";
        }
        operations.webJpa(dbms, database, user, password, INTERNALPATH, server,
            port, hibernateVersion);
        LOG.log(Level.INFO, "Web module persistence configured with JPA");
```

```
    }

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

        return operations.areModulesCreated();
    }

    /**
     * Configura el modulo web del proyecto que tiene el foco para el uso de los
     * servidores web Tomcat y Jetty.
     */
    @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() {

        operations.webDeploy(INTERNALPATH);
        LOG.log(Level.INFO, "Web application deployment configured");
    }
}
```

ProjectOps.java

```
package rooplusplus.roo.addon.project;

/**
 * Interface of commands that are available via the Roo shell.
 *
 * @since 1.1.1
 */
public interface ProjectOps {

    /**
```

```
     * Crea los diferentes modulos del proyecto Maven/Spring Roo que tiene el
     * foco.
     *
     * @param projectName Nombre del proyecto que tiene el foco
     * @param version Numero de version que se asignara a los diferentes modulos
     *        y al proyecto completo
     * @param internalPath Ruta interna, dentro del directorio "resources" del
     *        JAR, del directorio donde se encuentran realmente todos los
     *        resources.
     */
    void createModules(String projectName, String version, String internalPath);

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

    /**
     * Indica si ya se ha configurado la licencia en el proyecto que tiene el
     * foco.
     *
     * @return true si la licencia ya ha sido configurada; false en caso
     *         contrario
     */
    boolean isLicenseIntroduced();

    /**
     * Configura la licencia en el proyecto que tiene el foco.
     *
     * @param internalPath Ruta interna, dentro del directorio "resources" del
     *        JAR, del directorio donde se encuentran realmente todos los
     *        resources.
     */
    void license(String internalPath);
```

```
/**
 * Determina si ya se han ejecutado las acciones de configuracion sobre el
 * modulo web del proyecto que tiene el foco.
 *
 * @return true si ya se han ejecutado; false en caso contrario
 */
boolean isWebConfigured();

/**
 * Realiza diferentes acciones de configuracion sobre el modulo web del
 * proyecto que tiene el foco.
 *
 * @param internalPath Ruta interna, dentro del directorio "resources" del
 *        JAR, del directorio donde se encuentran realmente todos los
 *        resources.
 */
void webConfiguration(String internalPath);

/**
 * Determina si el modulo web del proyecto que tiene el foco ya ha sido
 * configurado para el uso de una base de datos.
 *
 * @return true si el modulo web ha sido configurado para el uso de una base
 *         de datos, false en caso contrario
 */
boolean isWebJpa();

/**
 * Configura el modulo "web" del proyecto que tiene el foco para el uso de
 * una base de datos.
 *
 * @param dbms Nombre del sistema de gestion de bases de datos: oracle,
 *        mysql, mssql, postgres
 * @param databaseName Nombre de la base de datos (que sera el nombre del
 *        proyecto si no se indica; no se utiliza en el caso de Oracle)
 * @param user Nombre de usuario para el acceso a la base de datos
 * @param password Password para el acceso a la base de datos
 * @param internalPath Ruta interna, dentro del directorio "resources" del
 *        JAR, del directorio donde se encuentran realmente todos los
```

```
     *        resources.
     * @param server Maquina que contiene la base de datos
     * @param port Puerto a traves del cual se accede a la base de datos
     * @param hibernateVersion Version de Hibernate a utilizar
     */
    void webJpa(String dbms, String databaseName, String user, String password,
                String internalPath, String server, String port,
                String hibernateVersion);

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

    /**
     * Configura el modulo web del proyecto que tiene el foco para el uso de los
     * servidores web Tomcat y Jetty.
     *
     * @param internalPath Ruta interna, dentro del directorio "resources" del
     *        JAR, del directorio donde se encuentran realmente todos los
     *        resources.
     */
    void webDeploy(String internalPath);
}
```

ProjectOpsImpl.java

```
package rooplusplus.roo.addon.project;

import java.io.File;
import java.util.Collection;
import java.util.Iterator;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.springframework.roo.addon.jpa.JpaOperations;
```

```
import org.springframework.roo.addon.web.mvc.jsp.JspOperations;
import org.springframework.roo.process.manager.FileManager;
import org.springframework.roo.project.MavenOperations;
import org.springframework.roo.project.Path;
import org.springframework.roo.project.PathResolver;
import org.springframework.roo.project.ProjectOperations;
import org.springframework.roo.project.maven.Pom;
import org.springframework.roo.project.packaging.PackagingProviderRegistry;

import rooplusplus.roo.addon.utils.CreateModules;
import rooplusplus.roo.addon.utils.FileUtilities;
import rooplusplus.roo.addon.utils.PomUtils;
import rooplusplus.roo.addon.utils.ProjectUtils;
import rooplusplus.roo.addon.utils.WebModuleUtils;


/**
 * Implementation of {@link ProjectOps} interface.
 *
 * @since 1.1.1
 */
@Component
@Service
public class ProjectOpsImpl
    implements ProjectOps {

    /**
     * Constructor de la clase.
     */
    public ProjectOpsImpl() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Cadena "web", usada en varias ocasiones en la clase.
     */
    private static final String WEB = "web";
```

```
/**
 * Cadena "pom.xml", usada en varias ocasiones en la clase.
 */
private static final String POM = "pom.xml";

/**
 * Cadena "/pom.xml", usada en varias ocasiones en la clase.
 */
private static final String BARRA_POM = "/pom.xml";

/**
 * Get a reference to the FileManager from the underlying OSGi container.
 * Make sure you are referencing the Roo bundle which contains this service
 * in your add-on pom.xml. Using the Roo file manager instead if
 * java.io.File gives you automatic rollback in case an Exception is thrown.
 */
@Reference
private FileManager fileManager;

/**
 * Get a reference to the ProjectOperations from the underlying OSGi
 * container. Make sure you are referencing the Roo bundle which contains
 * this service in your add-on pom.xml.
 */
@Reference
private ProjectOperations projectOperations;

/**
 * Referencia al MavenOperations del container OSGi.
 */
@Reference
private MavenOperations mavenOperations;

/**
 * Referencia al PackagingProviderRegistry del container OSGi.
 */
@Reference
private PackagingProviderRegistry packagingProviderRegistry;
```

```
/**
 * Referencia al JspOperations del container OSGi.
 */
@Reference
private JspOperations ops;

/**
 * Referencia al JpaOperations del container OSGi.
 */
@Reference
private JpaOperations jpaOperations;

@Override
public void createModules(String projectName, String version,
                          String internalPath) {

    // "Guardar" el modulo raiz para darle posteriormente el foco
    Pom raiz = projectOperations.getFocusedModule();
    PathResolver pathResolver =
        createTheModules(projectName, version, internalPath);
    // Paquete variables en los pom.xml
    String[] poms = {"", "core", "docs", "ear", "package", "scripts", WEB};
    CreateModules.replacePomVariables(poms, projectName, version,
        fileManager, pathResolver, projectOperations);
    WebModuleUtils.addSourcePlugin(
        new File(pathResolver.getFocusedIdentifier(Path.ROOT, POM))
            .getPath(), internalPath, this.getClass().getClassLoader());
    // Establecemos los valores correctos para el parent del proyecto
    // completo
    CreateModules.configureProjectParent(pathResolver, projectOperations);
    // Crear directorio "java" en el modulo web
    File java =
        new File(pathResolver.getFocusedIdentifier(Path.ROOT,
            "./src/main/java"));
    if (!java.mkdir()) {
        FileUtilities.problemCreatingDir(java.getAbsolutePath());
    }
    // Devolvemos el foco al modulo raiz
    projectOperations.setModule(raiz);
```

```
    // Establecer en 1.6 la version de Java en el pom del modulo web
    WebModuleUtils.setJavaVersion("1.6", projectOperations);
}

/**
 * Lleva a cabo la creacion de los modulos propiamente dicha.
 *
 * @param projectName Nombre del proyecto en el que se crearan los modulos
 * @param version Numero de version que se asignara a los diferentes modulos
 *        y al proyecto completo
 * @param internalPath Ruta interna, dentro del directorio "resources" del
 *        JAR, del directorio donde se encuentran realmente todos los
 *        resources.
 * @return Objeto que permite determinar que proyecto y modulo tienen el
 *         foco (especialmente necesario en esta situacion, en que el foco
 *         se mueve entre los modulos raiz y web de un proyecto)
 */
private PathResolver createTheModules(String projectName, String version,
                                      String internalPath) {

    CreateModules.createModule(WEB, projectName, version, fileManager,
        projectOperations, mavenOperations, packagingProviderRegistry);
    CreateModules.webSetupModule(WEB, projectName, fileManager, ops);
    // Crear el resto de modulos y realizar el resto de ajustes necesarios
    PathResolver pathResolver = projectOperations.getPathResolver();
    String path =
        (new File(pathResolver.getFocusedIdentifier(Path.ROOT, "..")))
            .getAbsolutePath();
    CreateModules.copyResources(projectName, path, internalPath, this
        .getClass().getClassLoader());
    return pathResolver;
}

@Override
public boolean areModulesCreated() {

    return ProjectUtils.areModulesCreated(projectOperations);
}
```

```
@Override
public boolean isLicenseIntroduced() {

    File f =
        new File(
            (ProjectUtils.getProjectPath(projectOperations) + BARRA_POM));
    if (!f.exists()) {
        return false;
    }
    String pom = FileUtilities.readFileAsString(f.getAbsolutePath());
    return pom.contains("license-maven-plugin");
}

@Override
public void license(String internalPath) {

    String pomPath =
        ProjectUtils.getProjectPath(projectOperations) + BARRA_POM;
    PomUtils.addLicensePlugin(pomPath);
    PomUtils.addLicense(pomPath);
    WebModuleUtils.addLicenseToWebModule(new File(pomPath).getParent(),
        internalPath, projectOperations.getFocusedProjectName(), this
            .getClass().getClassLoader());
}

@Override
public boolean isWebConfigured() {

    File f =
        WebModuleUtils.getWebModuleFile(
            "src/main/resources/META-INF/spring/application.properties",
            projectOperations);
    return f != null && f.exists();
}

@Override
public void webConfiguration(String internalPath) {

    String projectPath = ProjectUtils.getProjectPath(projectOperations);
```

```
        WebModuleUtils.copyApplicationPropertiesToWebModule(projectPath,
            internalPath, projectOperations.getFocusedProjectName(), this
                .getClass().getClassLoader());
        WebModuleUtils.updateAppContextWebMvcConfig(projectPath, internalPath,
            projectOperations.getFocusedProjectName(), this.getClass()
                .getClassLoader());
        WebModuleUtils.configureEclipsePlugin(projectPath,
            projectOperations.getFocusedProjectName());
    }

    @Override
    public boolean isWebJpa() {

        return WebModuleUtils.isWebJpa(projectOperations);
    }

    @Override
    public void webJpa(String dbms, String databaseName, String user,
                       String password, String internalPath, String server,
                       String port, String hibernateVersion) {

        String database = databaseName;
        // Si no se ha indicado el nombre de la base de datos asumimos que se
        // llama igual que el proyecto
        if ("".equals(database)) {
            database = projectOperations.getFocusedProjectName();
        }
        Pom raiz = projectOperations.getFocusedModule();
        String projectPath = ProjectUtils.getProjectPath(projectOperations);
        Collection<Pom> modules = projectOperations.getPoms();
        Iterator<Pom> i = modules.iterator();
        while (i.hasNext()) {
            Pom pom = i.next();
            if (pom.getModuleName().contains("-web")) {
                projectOperations.setModule(pom);
                CreateModules.jpaSetupModule(dbms, fileManager, jpaOperations,
                    projectOperations.getFocusedModuleName());
                break;
            }
```

```
        }
        projectOperations.setModule(raiz);
        callWebModuleUtils(projectPath, internalPath, dbms, server, database,
            user, password, port, hibernateVersion);
    }

    /**
     * Completa la configuracion del modulo "web" de un proyecto para el uso de
     * una base de datos, invocando a metodos de la clase WebModuleUtils.
     *
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param internalPath Ruta interna, dentro del directorio "resources" del
     *        JAR, del directorio donde se encuentran realmente todos los
     *        resources.
     * @param dbms Nombre del sistema de gestion de bases de datos: oracle,
     *        mysql, mssql, postgres
     * @param server Maquina que contiene la base de datos
     * @param database Nombre de la base de datos
     * @param user Nombre de usuario para el acceso a la base de datos
     * @param password Password para el acceso a la base de datos
     * @param port Puerto a traves del cual se accede a la base de datos
     * @param hibernateVersion Version de Hibernate a utilizar
     */
    private void callWebModuleUtils(String projectPath, String internalPath,
                                    String dbms, String server,
                                    String database, String user,
                                    String password, String port,
                                    String hibernateVersion) {

        // Introducir nuevas propiedades al final de database.properties, y
        // configurar la Base de Datos en el pom.xml del modulo web
        WebModuleUtils.copyDatabasePropertiesToWebModule(projectPath,
            internalPath, projectOperations.getFocusedProjectName(), database,
            this.getClass().getClassLoader());
        //Si se han especificado server o localhost como distintos de sus
        //valores por defecto, los ajustamos en database.properties
        if (!server.equals("localhost") || !port.equals("")) {
            WebModuleUtils.setDbPortServer(port, server, dbms, projectPath,
                projectOperations.getFocusedProjectName());
```

```
        }
        // Renombrar persistence.xml a persistenceInfo.xml, y modificar las
        // referencias a ese archivo de acuerdo con su nuevo nombre
        WebModuleUtils.renamePersistence(projectPath,
            projectOperations.getFocusedProjectName());
        // Copiar los ficheros y directorios de configuracion de Tomcat y Jetty
        WebModuleUtils.copyJettyTomcatConf(projectPath, internalPath, dbms,
            database, user, password, server, port, projectOperations
                .getFocusedProjectName(), this.getClass().getClassLoader());
        // Introducir en el modulo web los beans neceasarios para el uso de la
        // base de datos
        WebModuleUtils.updateAppContextDatabase(projectPath, internalPath,
            projectOperations.getFocusedProjectName(), this.getClass()
                .getClassLoader());
        // Establecer la version de Hibernate segun el parametro indicado
        WebModuleUtils.setHibernateVersion(hibernateVersion, projectOperations);
        // Introducir en el fichero web.xml la referencia al DataSource
        // correspondiente (necesario para que Jetty funcione correctamente)
        ProjectUtils.dataSourceWebXml(projectPath, internalPath,
            projectOperations, this.getClass().getClassLoader());
    }

    @Override
    public boolean isWebDeployed() {

        File f = WebModuleUtils.getWebModuleFile(POM, projectOperations);
        if (f == null || !f.exists()) {
            return false;
        }
        String pom = FileUtilities.readFileAsString(f.getAbsolutePath());
        return pom
            .contains("<jettyConfig>src/main/jetty/jetty.xml</jettyConfig>");
    }

    @Override
    public void webDeploy(String internalPath) {

        // Si ya se habia hecho anteriormente web deploy y en ningun momento se
        // hizo web jpa salimos (no hay nada que hacer)
```

```
        if (isWebDeployed() && !isWebJpa()) {
            return;
        }
        if (isWebDeployed()) {
            // Deshacer las partes del web deploy anterior que puedan traer
            // problemas
            ProjectUtils.undoWebDeploy(projectOperations);
        }
        ProjectUtils.setMavenPlugins(projectOperations);
        if (!isWebJpa()) {
            // Copiamos los ficheros de configuracion de Jetty
            WebModuleUtils.copyJettyConfWithoutDatabase(ProjectUtils
                .getProjectPath(projectOperations), projectOperations
                .getFocusedProjectName(), internalPath, this.getClass()
                .getClassLoader());
        }
    }
}
```

package-info.java

```
/**
 * Paquete para la creacion y configuracion de proyectos Spring Roo multimodulo.
 */
package rooplusplus.roo.addon.project;
```

Paquete Usermgmt

UsermgmtCommands.java

```
package rooplusplus.roo.addon.usermgmt;

import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
import org.apache.felix.scr.annotations.Service;
import org.osgi.service.component.ComponentContext;
import org.springframework.roo.shell.CliAvailabilityIndicator;
import org.springframework.roo.shell.CliCommand;
import org.springframework.roo.shell.CommandMarker;

/**
 * Comandos relacionados con la configuracion de la gestion de usuarios en los
 * proyectos. The command class is registered by the Roo shell following an
 * automatic classpath scan. You can provide simple user presentation-related
 * logic in this class. You can return any objects from each method, or use the
 * logger directly if you'd like to emit messages of different severity (and
 * therefore different colors on non-Windows systems).
 *
 * @since 1.1.1
 */
@Component
// Use these Apache Felix annotations to register your commands class in the Roo
// container
@Service
public class UsermgmtCommands
    implements CommandMarker { // All command types must implement the

    // CommandMarker interface

    /**
     * Constructor de la clase.
     */
```

```
public UsermgmtCommands() {

    // Todas las clases deben tener constructor.
}

/**
 * Get hold of a JDK Logger.
 */
private static final Logger LOG = Logger
    .getLogger("UsermgmtCommands.class");

/**
 * Ruta interna, dentro del directorio "resources" del JAR, del directorio
 * donde se encuentran realmente todos los resources.
 */
private static final String INTERNALPATH = "rooAddon/";

/**
 * Get a reference to the AddonOperations from the underlying OSGi
 * container.
 */
@Reference
private UsermgmtOperations operations;

/**
 * The activate method for this OSGi component, this will be called by the
 * OSGi container upon bundle activation (result of the 'addon install'
 * command).
 *
 * @param context the component context can be used to get access to the
 *        OSGi container (ie find out if certain bundles are active)
 */
protected void activate(ComponentContext context) {

    // Es necesario que exista este metodo
}

/**
 * The deactivate method for this OSGi component, this will be called by the
```

```
     * OSGi container upon bundle deactivation (result of the 'addon remove'
     * command).
     *
     * @param context the component context can be used to get access to the
     *        OSGi container (ie find out if certain bundles are active)
     */
    protected void deactivate(ComponentContext context) {

        // Es necesario que exista este metodo
    }

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

        return operations.canUsermgmtSetup();
    }

    /**
     * Configura el modulo web del proyecto que tiene el foco para el uso de un
     * servicio de autenticacion de usuarios, sea mediante un modulo externo o
     * mediante el uso de las librerias de Spring Security, segun lo que haya
     * configurado el usuario.
     */
    @CliCommand(value = "roo++ usermgmt setup", help = "Configures the project
to use an authentication server")
    public void usermgmtSetup() {

        operations.usermgmtSetup(INTERNALPATH);
        LOG.log(Level.INFO, "User management configured");
    }

}
```

UsermgmtOperations.java

```
package rooplusplus.roo.addon.usermgmt;

/**
 * Interface of commands that are available via the Roo shell.
 *
 * @since 1.1.1
 */
public interface UsermgmtOperations {

    /**
     * Configura el modulo web del proyecto que tiene el foco para el uso de un
     * servicio de autenticacion de usuarios, sea mediante un modulo externo o
     * mediante el uso de las librerias de Spring Security, segun lo que haya
     * configurado el usuario.
     *
     * @param internalPath Ruta interna, dentro del directorio "resources" del
     *        JAR, del directorio donde se encuentran realmente todos los
     *        resources.
     */
    void usermgmtSetup(String internalPath);

    /**
     * Indica si es posible ejecutar el comando usermgmt setup en el proyecto
     * que tiene el foco.
     *
     * @return true si es posible ejecutar el comando en este momento; false en
     *         caso contrario
     */
    boolean canUsermgmtSetup();
}
```

UsermgmtOperationsImpl.java

```
package rooplusplus.roo.addon.usermgmt;

import java.io.File;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Reference;
```

```
import org.apache.felix.scr.annotations.Service;
import org.springframework.roo.project.ProjectOperations;

import rooplusplus.roo.addon.utils.FileUtilities;
import rooplusplus.roo.addon.utils.ProjectUtils;
import rooplusplus.roo.addon.utils.UsermgmtUtils;
import rooplusplus.roo.addon.utils.WebModuleUtils;


/**
 * Implementation of {@link UsermgmtOperations} interface.
 *
 * @since 1.1.1
 */
@Component
@Service
public class UsermgmtOperationsImpl
    implements UsermgmtOperations {

    /**
     * Constructor de la clase.
     */
    public UsermgmtOperationsImpl() {

        // Todas las clases deben tener constructor.
    }

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String USERMGMT = "usermgmt";

    /**
     * Get a reference to the ProjectOperations from the underlying OSGi
     * container. Make sure you are referencing the Roo bundle which contains
     * this service in your add-on pom.xml.
     */
    @Reference
    private ProjectOperations projectOperations;
```

```
    @Override
    public void usermgmtSetup(String internalPath) {

        String projectPath = ProjectUtils.getProjectPath(projectOperations);
        if (new File(FileUtilities.getConfigurationFilePath(USERMGMT)).exists())
{
            // Caso de que el usuario haya definido su propio modulo de gestion
            // de usuarios
            UsermgmtUtils.usermgmtSetupCustom(internalPath, projectPath,
                projectOperations, this.getClass().getClassLoader());
        } else {
            // Usamos las librerias de Spring para la gestion de usuarios
            UsermgmtUtils.usermgmtSetupSpring(internalPath, projectPath,
                projectOperations, this.getClass().getClassLoader());
        }
    }

    @Override
    public boolean canUsermgmtSetup() {

        boolean platformConsigurationOk = true;
        // Si se ha de usar la plataforma, tiene que haberse ejecutado platform
        // configuration
        if (new File(FileUtilities.getConfigurationFilePath(USERMGMT)).exists()
            && !WebModuleUtils
                .getWebModuleFile(
                    "src/main/resources/META-INF/spring/applicationContext-
platform.xml",
                    projectOperations).exists()) {
            platformConsigurationOk = false;
        }
        // Si aun no se han creado los modulos o si ya se ha hecho usermgmt
        // setup, o si aun no se ha hecho web configuration (y por tanto no
        // existe application.properties) no se puede ejecutar el comando
        return (ProjectUtils.areModulesCreated(projectOperations)
            && !WebModuleUtils
                .getWebModuleFile(
                    "src/main/resources/META-INF/spring/applicationContext-
security.xml",
```

```
                    projectOperations).exists()
            && WebModuleUtils.getWebModuleFile(
                "src/main/resources/META-INF/spring/application.properties",
                projectOperations).exists() && platformConsigurationOk);
    }
}
```

package-info.java

```
/**
 * Paquete para la configuracion de la gestion de usuarios.
 */
package rooplusplus.roo.addon.usermgmt;
```

Paquete Utils

CreateModules.java

```
package rooplusplus.roo.addon.utils;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.apache.commons.io.FileUtils;
import org.springframework.roo.addon.jpa.JdbcDatabase;
import org.springframework.roo.addon.jpa.JpaOperations;
import org.springframework.roo.addon.jpa.OrmProvider;
import org.springframework.roo.addon.web.mvc.jsp.JspOperations;
import org.springframework.roo.process.manager.FileManager;
import org.springframework.roo.project.GAV;
import org.springframework.roo.project.MavenOperations;
import org.springframework.roo.project.Path;
import org.springframework.roo.project.PathResolver;
import org.springframework.roo.project.ProjectOperations;
import org.springframework.roo.project.packaging.PackagingProvider;
import org.springframework.roo.project.packaging.PackagingProviderRegistry;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.xml.sax.SAXException;

/**
 * Utilidades para la creacion de los diferentes modulos de la aplicacion.
 *
```

```
 * @author javier.j.menendez
 */
public final class CreateModules {

    /**
     * Constructor privado para que esta clase de utilidades no pueda ser
     * instanciada.
     */
    private CreateModules() {

    }

    /**
     * Log para mostrar informacion al usuario.
     */
    private static final Logger LOG = Logger.getLogger("CreateModules.class");

    /**
     * Numero de version principal de Java a partir de la cual funcionaran los
     * modulos creados.
     */
    private static final int JAVA_VERSION = 6;

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String PARENT_POM = "../pom.xml";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String DOCS_POM = "-docs/pom.xml";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String GUION = "-";

    /**
```

```
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String UTF8 = "UTF-8";

    /**
     * Cadena con el caracter "/", usado en varias ocasiones en la clase.
     */
    private static final String BARRA = "/";

    /**
     * Copia los resources indicados en los ficheros dirs.txt y textfiles.txt,
     * que contienen los directorios, y archivos de texto, respectivamente, que
     * seran 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).
     *
     * @param projectName nombre del proyecto
     * @param path path del directorio en que se encuentra ubicado el proyecto
     * @param orgPath path interno en el que se encuentran los resources de este
     *        proyecto, dentro del directorio de este nombre (acabado en "/")
     * @param cl ClassLoader correspondiente al hilo en que se produce la
     *        ejecucion
     */
    public static void copyResources(String projectName, String path,
                                     String orgPath, ClassLoader cl) {

        try {
            FileUtils.deleteDirectory(new File(path + "/src"));
            createDirs(path, projectName, orgPath, cl);
            copyTextFiles(path, projectName, orgPath, cl);
            copyDocFiles(path, projectName);
            addDocsUserPlugins(path, projectName);
            addDocbkxPluginFonts(path, projectName);
        } catch (IOException ex) {
            docsModuleConfProblem();
        } catch (XPathExpressionException e) {
            docsModuleConfProblem();
        } catch (ParserConfigurationException e) {
            docsModuleConfProblem();
```

```
        } catch (SAXException e) {
            docsModuleConfProblem();
        } catch (TransformerException e) {
            docsModuleConfProblem();
        }
    }

    /**
     * Informa al usuario de que se ha producido un problema con la
     * configuracion del modulo "docs" del proyecto.
     */
    private static void docsModuleConfProblem() {

        LOG.log(Level.SEVERE,
            "ERROR. Problem with user configuration files for \"docs\" module");
    }

    /**
     * Establece, en los ficheros pom.xml indicados, los groupId, artifactId y
     * version, con sus valores correspondientes, incluyendo estos mismos
     * atributos para el parent del modulo, salvo el caso del pom raiz. El foco
     * ha de estar puesto en el modulo web para que este metodo funcione de la
     * forma prevista.
     *
     * @param poms Array con los nombres de los diferentes modulos, mas uno
     *        vacio correspondiente a la raiz del proyecto. Las variables seran
     *        reemplazadas en cada elemento de este array
     * @param projectName Nombre del proyecto
     * @param version Version del proyecto
     * @param fileManager Objeto FileManager procedente de la clase que invoca a
     *        este metodo
     * @param pathResolver Objeto PathResolver procedente de la clase que invoca
     *        a este metodo
     * @param projectOperations Objeto ProjectOperations procedente de la clase
     *        que invoca a este metodo
     */
    public static void replacePomVariables(String[] poms, String projectName,
                                           String version,
                                           FileManager fileManager,
```

```
                                        PathResolver pathResolver,
                                        ProjectOperations projectOperations)
{

        String pomPath, moduleName;
        for (int i = 0; i < poms.length; i++) {
            String[] results = trataPom(poms[i], projectName, pathResolver);
            pomPath = results[0];
            moduleName = results[1];
            PomUtils.replaceVariables(pomPath, moduleName, projectName,
                version, projectOperations.getFocusedTopLevelPackage()
                    .getFullyQualifiedPackageName());
            if ("web".equals(poms[i])) {
                // Poner version y dependencias en el modulo web, que fue creado
                // por Spring Roo en lugar de copiando los archivos
                configureWebModule(pomPath, projectName, version, fileManager,
                    projectOperations);
            }
        }
    }

    /**
     * Obtiene el la ruta del fichero pom.xml de un modulo del proyecto, y el
     * nombre de dicho modulo.
     *
     * @param pom Nombre del modulo, o cadena vacia si el modulo es el raiz
     * @param projectName Nombre del proyecto
     * @param pathResolver Objeto PathResolver procedente de la clase que invoca
     *        a replacePomVariables
     * @return Ruta del pom.xml y nombre del modulo (si es el raiz devuelve el
     *         nombre del proyecto)
     */
    private static String[] trataPom(String pom, String projectName,
                                     PathResolver pathResolver) {

        String pomPath, moduleName;
        if (pom.isEmpty()) {
            pomPath =
                (new File(pathResolver.getFocusedIdentifier(Path.ROOT,
                    PARENT_POM))).getPath();
```

```
            moduleName = projectName;
        } else {
            pomPath =
                (new File(pathResolver.getFocusedIdentifier(Path.ROOT, "../"
                    + projectName + GUION + pom + "/pom.xml"))).getPath();
            moduleName = pom;
            LOG.log(Level.INFO, "Module created: " + moduleName);
        }
        return new String[] {pomPath, moduleName};
    }

    /**
     * Pone la version y las dependencias en el modulo web de la aplicacion, que
     * fue creado por Spring Roo en lugar de copiando los archivos desde los
     * resources del JAR.
     *
     * @param pomPath Ruta del fichero pom.xml del modulo web de la aplicacion
     * @param projectName Nombre del proyecto
     * @param version Version del proyecto, y por tanto tambien del modulo web
     * @param fileManager Objeto FileManager procedente de la clase que invoca a
     *        replacePomVariables
     * @param projectOperations Objeto ProjectOperations procedente de la clase
     *        que invoca a replacePomVariables
     */
    private static void configureWebModule(String pomPath, String projectName,
                                           String version,
                                           FileManager fileManager,
                                           ProjectOperations projectOperations)
{

        PomUtils.setVersion(pomPath, version, fileManager);
        try {
            PomUtils.addDependency(pomPath, projectOperations
                .getFocusedTopLevelPackage().getFullyQualifiedPackageName(),
                projectName + GUION + "core", version);
        } catch (IOException ex) {
            pomWrongFormat(pomPath);
        } catch (XPathExpressionException e) {
            pomWrongFormat(pomPath);
        } catch (ParserConfigurationException e) {
```

```
            pomWrongFormat(pomPath);
        } catch (SAXException e) {
            pomWrongFormat(pomPath);
        } catch (TransformerException e) {
            pomWrongFormat(pomPath);
        }
    }

    /**
     * Informa al usuario de que un fichero pom.xml no tiene el formato
     * adecuado.
     *
     * @param pomPath Ruta del fichero pom.xml
     */
    private static void pomWrongFormat(String pomPath) {

        LOG.log(Level.SEVERE, "ERROR. File \"" + pomPath + "has wrong format");
    }

    /**
     * Establece en el fichero pom.xml de la raiz del proyecto el groupId,
     * artifactId y version del parent del mismo, siempre que exista, segun los
     * valores configurados por el usuario El foco ha de estar puesto en el
     * modulo web para que este metodo funcione de la forma prevista.
     *
     * @param pathResolver Objeto PathResolver procedente de la clase que invoca
     *        a este metodo
     * @param projectOperations Objeto ProjectOperations procedente de la clase
     *        que invoca a este metodo
     */
    public static void configureProjectParent(PathResolver pathResolver,
                                              ProjectOperations
projectOperations) {

        // Establecemos los valores correctos para el parent del proyecto
        // completo
        String pomPath =
            (new File(pathResolver.getFocusedIdentifier(Path.ROOT, PARENT_POM)))
                .getPath();
        BufferedReader confFile = null;
```

```
        try {
            String confFilePath =
                FileUtilities.getConfigurationFilePath("parent.txt");
            confFile = new BufferedReader(new FileReader(confFilePath));
            String groupId = confFile.readLine();
            String artifactId = confFile.readLine();
            String ver = confFile.readLine();
            if (groupId == null || artifactId == null || ver == null) {
                throw new IOException();
            }
            FileUtilities.replaceText("groupId.groupId.groupId", groupId,
                pomPath);
            FileUtilities.replaceText("artifactIdartifactId", artifactId,
                pomPath);
            FileUtilities.replaceText("versionversion", ver, pomPath);
        } catch (IOException ex) {
            String warning =
                "WARNING. Configuration file parent.txt not found or without
correct content. The project won't have parent";
            LOG.log(Level.WARNING, warning);
            // Si el proyecto no tiene parent eliminamos ese campo del pom.xml
            // raiz
            XmlUtils.eliminaNodoSiExiste(pomPath, "/project/parent");
        } finally {
            try {
                if (confFile != null) {
                    confFile.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Introduce en el pom.xml del modulo docs la referencia a las fuentes que
     * el usuario haya indicado en el fichero de configuracion correspondiente,
     * para generar la documentacion del proyecto mediante DocBook XSL.
     *
     * @param path Ruta completa del directorio raiz del proyecto
```

```
     * @param projectName Nombre del proyecto
     * @throws IOException Si el pom.xml del modulo docs no existe o no tiene el
     *         formato adecuado, o si ocurre lo mismo con el fichero de
     *         configuracion fonts.xml
     * @throws SAXException Si el pom.xml del modulo docs no existe o no tiene
     *         el formato adecuado, o si ocurre lo mismo con el fichero de
     *         configuracion fonts.xml
     * @throws ParserConfigurationException Si el pom.xml del modulo docs no
     *         existe o no tiene el formato adecuado, o si ocurre lo mismo con
     *         el fichero de configuracion fonts.xml
     * @throws XPathExpressionException Si el pom.xml del modulo docs no existe
     *         o no tiene el formato adecuado, o si ocurre lo mismo con el
     *         fichero de configuracion fonts.xml
     * @throws TransformerException Si el pom.xml del modulo docs no existe o no
     *         tiene el formato adecuado, o si ocurre lo mismo con el fichero de
     *         configuracion fonts.xml
     */
    private static void addDocbkxPluginFonts(String path, String projectName)
        throws ParserConfigurationException,
            SAXException,
            IOException,
            XPathExpressionException,
            TransformerException {

        Document doc =
            XmlUtils.createDocumentFromFile(path + BARRA + projectName
                + DOCS_POM);
        Node docbkxConf =
            XmlUtils
                .evaluateXPath(

"/project/build/plugins/plugin/executions/execution/configuration",
                    doc).item(0);
        Document fontsDoc =
            XmlUtils.createDocumentFromFile(FileUtilities
                .getConfigurationFilePath("doc/fonts.xml"));
        Node fonts =
            doc.adoptNode(fontsDoc.getDocumentElement().cloneNode(true));
        docbkxConf.appendChild(fonts);
        // Guardar los cambios
```

```
        XmlUtils.guardaXml(doc, path + BARRA + projectName + DOCS_POM);
    }

    /**
     * Introduce en el pom.xml del modulo docs los plugins que el usuario haya
     * colocado en el directorio de configuracion correspondiente, para la
     * generacion de los PDF de documentacion del proyecto, y establece como
     * destino target en lugar de target/help-pdf en caso de que no se haya
     * indicado ningun plugin, ya que estos ultimos son los responsables de
     * colocar los PDF finales con la documentacion en su destino final, el
     * directorio target.
     *
     * @param path Ruta completa del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     * @throws IOException Si el pom.xml del modulo docs no existe o no tiene el
     *         formato adecuado, o si alguno de los ficheros en el directorio de
     *         configuracion de plugins no tiene un formato valido, o si no
     *         existe dicho directorio
     * @throws SAXException Si el pom.xml del modulo docs no existe o no tiene
     *         el formato adecuado, o si alguno de los ficheros en el directorio
     *         de configuracion de plugins no tiene un formato valido, o si no
     *         existe dicho directorio
     * @throws ParserConfigurationException Si el pom.xml del modulo docs no
     *         existe o no tiene el formato adecuado, o si alguno de los
     *         ficheros en el directorio de configuracion de plugins no tiene un
     *         formato valido, o si no existe dicho directorio
     * @throws XPathExpressionException Si el pom.xml del modulo docs no existe
     *         o no tiene el formato adecuado, o si alguno de los ficheros en el
     *         directorio de configuracion de plugins no tiene un formato
     *         valido, o si no existe dicho directorio
     * @throws TransformerException Si el pom.xml del modulo docs no existe o no
     *         tiene el formato adecuado, o si alguno de los ficheros en el
     *         directorio de configuracion de plugins no tiene un formato
     *         valido, o si no existe dicho directorio
     */
    private static void addDocsUserPlugins(String path, String projectName)
        throws ParserConfigurationException,
            SAXException,
            IOException,
```

```
            XPathExpressionException,
            TransformerException {

        Document doc =
            XmlUtils.createDocumentFromFile(path + BARRA + projectName
                + DOCS_POM);
        Node plugins =
            XmlUtils.evaluateXPath("/project/build/plugins", doc).item(0);
        Document pluginDoc;
        Node plugin;
        File confDir =
            new File(FileUtilities.getConfigurationFilePath("docPlugins"));
        File[] pluginsList = confDir.listFiles();
        for (int i = 0; i < pluginsList.length; i++) {
            pluginDoc =
                XmlUtils.createDocumentFromFile(pluginsList[i]
                    .getAbsolutePath());
            plugin = pluginDoc.getDocumentElement().cloneNode(true);
            plugins.appendChild(doc.adoptNode(plugin));
        }
        // Guardar los cambios
        XmlUtils.guardaXml(doc, path + BARRA + projectName + DOCS_POM);
        // Si no se utiliza ningun plugin adicional los PDF iran directamente al
        // target
        if (pluginsList.length == 0) {
            FileUtilities.replaceText("target/help-pdf", "target", path + BARRA
                + projectName + DOCS_POM);
        }
    }

    /**
     * Crea los diferentes directorios del proyecto (excluyendo el modulo web),
     * segun lo indicado fichero dirs.txt, situado en la raiz del directorio
     * resources del JAR.
     *
     * @param path Ruta completa del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     * @param orgPath Ruta interna de los resources del JAR
     * @param cl ClassLoader de la clase que invoca a este metodo, necesario `
```

```
     *        para obtener los resources del JAR
     * @throws IOException Si no existiera el fichero dirs.txt en el JAR (algo
     *         que en teoria nunca deberia ocurrir), o si no fuera posible crear
     *         alguno de los directorios
     */
    private static void createDirs(String path, String projectName,
                                   String orgPath, ClassLoader cl)
        throws IOException {

        BufferedReader in = null;
        try {
            // Leer dirs
            ArrayList<String> dirs = new ArrayList<String>();
            in =
                new BufferedReader(new InputStreamReader(
                    cl.getResourceAsStream(orgPath + "dirs.txt"), UTF8));
            String str;
            while ((str = in.readLine()) != null) {
                dirs.add(str);
            }
            // Crear los directorios
            File dir;
            for (int i = 0; i < dirs.size(); i++) {
                dir =
                    new File(path + BARRA + projectName + GUION + dirs.get(i));
                if (!dir.mkdir()) {
                    FileUtilities.problemCreatingDir(dir.getAbsolutePath());
                }
            }
        } finally {
            try {
                if (in != null) {
                    in.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }
```

```
/**
 * Copia al proyecto que esta siendo creado los ficheros de texto basicos
 * que siempre son necesarios (excepto los del modulo web), desde el
 * directorio resources del JAR, segun lo indicado en textiles.txt, en la
 * raiz de dichos resources.
 *
 * @param path Ruta completa del directorio raiz del proyecto
 * @param projectName Nombre del proyecto
 * @param orgPath Ruta interna de los resources del JAR
 * @param cl ClassLoader de la clase que invoca a este metodo, necesario `
 *        para obtener los resources del JAR
 * @throws IOException Si no existiera el fichero textfiles.txt en el JAR
 *         (algo que en teoria nunca deberia ocurrir), o si no fuera posible
 *         copiar alguno de los ficheros de texto
 */
private static void copyTextFiles(String path, String projectName,
                                  String orgPath, ClassLoader cl)
    throws IOException {

    BufferedReader in = null;
    try {
        // Leer textfiles
        ArrayList<String> textFiles = new ArrayList<String>();
        in =
            new BufferedReader(new InputStreamReader(
                cl.getResourceAsStream(orgPath + "textfiles.txt"), UTF8));
        String str;
        while ((str = in.readLine()) != null) {
            textFiles.add(str);
        }
        // Copiar los archivos de texto indicados
        for (int i = 0; i < textFiles.size(); i++) {
            ProjectUtils.copyTextFile(path, projectName, textFiles.get(i),
                orgPath, cl);
        }
    } finally {
        try {
            if (in != null) {
```

```
                in.close();
            }
        } catch (IOException ex) {
            FileUtilities.problemClosingFile();
        }
    }
}

/**
 * Copia al modulo doc del proyecto que esta siendo creado los ficheros
 * necesarios (tales como fuentes, PDF, o XSL), todos ellos desde el
 * directorio de configuracion correspondiente.
 *
 * @param path Ruta completa del directorio raiz del proyecto
 * @param projectName Nombre del proyecto
 * @throws IOException Si no existe el directorio de configuracion
 *         necesario, o si no se puede copiar su contenido
 */
private static void copyDocFiles(String path, String projectName)
    throws IOException {

    // Copiar los archivos del modulo doc dependientes del usuario
    File confDir = new File(FileUtilities.getConfigurationFilePath("doc"));
    File[] docDirs = confDir.listFiles();
    for (int i = 0; i < docDirs.length; i++) {
        if (docDirs[i].isDirectory()) {
            FileUtils.copyDirectory(docDirs[i], new File(path + BARRA
                + projectName + "-docs/src/" + docDirs[i].getName()));
        }
    }
}

/**
 * Crea un modulo con el nombre indicado, invocando a Spring Roo.
 *
 * @param moduleName Nombre del modulo a crear
 * @param projectName Nombre del proyecto al que pertenecera dicho modulo
 * @param version Nombre (generalmente un numero) de la version del modulo a
 *        crear
```

```
     * @param fileManager Objeto FileManager de la clase que invoca al metodo,
     *        necesario para forzar determinadas operaciones de Entrada/Salida,
     *        para asegurarse de que todos los cambios realmente han sido
     *        guardados en disco
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo necesario para conocer que paquete tiene el foco
     * @param mavenOperations Objeto MavenOperations que sera utilizado para la
     *        creacion del modulo propiamente dicha
     * @param packagingProviderRegistry Objeto PackagingProviderRegistry, para
     *        obtener el tipo de empaquetamiento jar
     */
    public static void createModule(String moduleName,
                                    String projectName,
                                    String version,
                                    FileManager fileManager,
                                    ProjectOperations projectOperations,
                                    MavenOperations mavenOperations,
                                    PackagingProviderRegistry
packagingProviderRegistry) {

        fileManager.commit();
        GAV parentPom =
            new GAV(projectOperations.getFocusedTopLevelPackage()
                .getFullyQualifiedPackageName(), projectName, version);
        PackagingProvider packagingProvider =
            packagingProviderRegistry.getPackagingProvider("jar");
        mavenOperations.createModule(
            projectOperations.getFocusedTopLevelPackage(), parentPom,
            projectName + GUION + moduleName, packagingProvider, JAVA_VERSION,
            projectName + GUION + moduleName);
        fileManager.commit();
    }

    /**
     * Configura un modulo para el uso de una base de datos mediante JPA.
     *
     * @param database Nombre del sistema de gestion de bases de datos a
     *        utilizar (mssql, mysql, postgres, u oracle; este ultimo tambien se
     *        utilizara si se indica cualquier otro valor)
     * @param fileManager Objeto FileManager, necesario para forzar determinadas
```

```
     *        operaciones de Entrada/Salida, para asegurarse de que todos los
     *        cambios realmente han sido guardados en disco.
     * @param jpaOperations Objeto JpaOperations de la clase que invoca al
     *        metodo, necesario para configurar el acceso del modulo a la base
     *        de datos
     * @param moduleName Nombre del modulo que se conectara a la base de datos
     *        mediante JPA
     */
    public static void jpaSetupModule(String database, FileManager fileManager,
                                      JpaOperations jpaOperations,
                                      String moduleName) {

        fileManager.commit();
        JdbcDatabase db;
        if ("mssql".equals(database)) {
            db = JdbcDatabase.MSSQL;
        } else if ("mysql".equals(database)) {
            db = JdbcDatabase.MYSQL;
        } else if ("postgres".equals(database)) {
            db = JdbcDatabase.POSTGRES;
        } else {
            db = JdbcDatabase.ORACLE;
        }
        jpaOperations.configureJpa(OrmProvider.HIBERNATE, db, null, null, null,
            null, null, null, null, null, moduleName);
        fileManager.commit();
    }

    /**
     * Configura un modulo para que ofrezca una interfaz web.
     *
     * @param moduleName Nombre del modulo que tendra una interfaz web
     * @param projectName Nombre del proyecto al que pertenece dicho modulo
     * @param fileManager Objeto FileManager, necesario para forzar determinadas
     *        operaciones de Entrada/Salida, para asegurarse de que todos los
     *        cambios realmente han sido guardados en disco.
     * @param ops Objeto JspOperations de la clase que invoca al metodo, que
     *        sera utilizado para crear la interfaz web del modulo
     */
```

```
    public static void webSetupModule(String moduleName, String projectName,
                                       FileManager fileManager, JspOperations
ops) {

        fileManager.commit();
        ops.installCommonViewArtefacts(projectName + GUION + moduleName);
        fileManager.commit();
    }
}
```

EarPackaging.java

```
package rooplusplus.roo.addon.utils;

import java.util.Collection;

import org.apache.felix.scr.annotations.Component;
import org.apache.felix.scr.annotations.Service;
import org.springframework.roo.project.Path;
import org.springframework.roo.project.packaging.AbstractPackagingProvider;

/**
 * Ear Packaging.
 */
@Component
@Service
public class EarPackaging
    extends AbstractPackagingProvider {

    /**
     * Declaracion de un packaging de nombre ear, no soportado nativamente por
     * Roo.
     */
    public static final String NAME = "ear";

    /**
     * Constructor invoked by the OSGi container.
     */
    public EarPackaging() {

        super(NAME, NAME, "jar-pom-template.xml");
```

```
    }

    @Override
    public Collection<Path> getPaths() {

        return null;
    }

    @Override
    public boolean isDefault() {

        return false;
    }
}
```

FileUtilities.java

```
package rooplusplus.roo.addon.utils;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.apache.commons.io.FileUtils;

/**
 * Utilidades para la gestion de ficheros.
 *
 * @author javier.j.menendez
 */
```

```
public final class FileUtilities {

    /**
     * Constructor privado para que esta clase de utilidades no pueda ser
     * instanciada.
     */
    private FileUtilities() {

    }

    /**
     * Logger para informar al usuario de diferentes eventos.
     */
    private static final Logger LOG = Logger.getLogger("FileUtilities.class");

    /**
     * Cadena con el caracter "/", usado en varias ocasiones en la clase.
     */
    private static final String BARRA = "/";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String FILE_ENCODING = "file.encoding";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String UTF8 = "UTF-8";

    /**
     * "Numero magico" utilizado como size del buffer en la lectura de ficheros.
     */
    private static final int MIL = 1000;

    /**
     * "Numero magico" utilizado como size del buffer en la lectura de ficheros.
     */
    private static final int BIN_KILO = 1024;
```

```
/**
 * Lee un fichero de texto, devolviendo su contenido en un String, y
 * lanzando una excepcion si el fichero indicado no existe.
 *
 * @param filePath Ruta completa del fichero a leer
 * @return Contenido del fichero indicado
 * @throws IOException Si el fichero indicado no existe
 */
public static String readFileAsStringThrows(String filePath)
    throws IOException {

    BufferedReader reader = null;
    try {
        StringBuffer fileData = new StringBuffer(MIL);
        reader = new BufferedReader(new FileReader(filePath));
        char[] buf = new char[BIN_KILO];
        int numRead = 0;
        while ((numRead = reader.read(buf)) != -1) {
            String readData = String.valueOf(buf, 0, numRead);
            fileData.append(readData);
            buf = new char[BIN_KILO];
        }
        return fileData.toString();
    } finally {
        try {
            if (reader != null) {
                reader.close();
            }
        } catch (IOException ex) {
            FileUtilities.problemClosingFile();
        }
    }
}

/**
 * Lee un fichero de texto, devolviendo su contenido en un String, o la
 * cadena vacia si el fichero indicado no existe.
 *
```

```
     * @param filePath Ruta completa del fichero a leer
     * @return Contenido del fichero indicado
     */
    public static String readFileAsString(String filePath) {

        BufferedReader reader = null;
        try {
            StringBuffer fileData = new StringBuffer(MIL);
            reader = new BufferedReader(new FileReader(filePath));
            char[] buf = new char[BIN_KILO];
            int numRead = 0;
            while ((numRead = reader.read(buf)) != -1) {
                String readData = String.valueOf(buf, 0, numRead);
                fileData.append(readData);
                buf = new char[BIN_KILO];
            }
            return fileData.toString();
        } catch (IOException ex) {
            return "";
        } finally {
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Reemplaza en un fichero de texto todas las ocurrencias de una cadena por
     * otra.
     *
     * @param oldText Cadena a Paquete
     * @param newText Cadena que Paquetea a la anterior
     * @param filePath Ruta completa del fichero de texto en el que se
     *        Paquetean las cadenas
     */
```

```
public static void replaceText(String oldText, String newText,
                               String filePath) {

    PrintWriter out = null;
    try {
        String file = readFileAsString(filePath);
        file = file.replace(oldText, newText);
        out =
            new PrintWriter(new BufferedWriter(new FileWriter(filePath,
                false)));
        out.print(file);
    } catch (IOException ex) {
        LOG.log(Level.SEVERE, "ERROR. \"" + filePath
            + "\" file not found or has wrong content");
    } finally {
        if (out != null) {
            out.close();
        }
    }
}

/**
 * Devuelve la ruta completa de un archivo o directorio de configuracion.
 * 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 esta no esta definida, contenida en el
 * directorio "home" del usuario.
 *
 * @param internalFilePath Ruta del archivo o directorio de configuracion,
 *        tomando como raiz el directorio "RooAddonsData".
 * @return Ruta completa del archivo o directorio de configuracion
 */
public static String getConfigurationFilePath(String internalFilePath) {

    // Si la variable de entorno "ROOADDON" esta definida (DEBE ACABAR EN
    // "/"), se sacara de alli; si no, de "RooAddonsData" en el home del
    // usuario
    String env = System.getenv("ROOADDON");
    if (env == null) {
```

```
            return System.getProperty("user.home") + "/RooAddonsData/"
                + internalFilePath;
        } else {
            return env + internalFilePath;
        }
    }

    /**
     * Elimina todas las lineas vacias de un fichero de texto, es decir, todas
     * las lineas que no tengan ningun caracter o que solo tengan tabuladores
     * y/o espacios en blanco.
     *
     * @param path Ruta completa del fichero de texto del que se eliminaran las
     *        lineas vacias
     */
    public static void removeBlankLines(String path) {

        BufferedReader br = null;
        PrintWriter out = null;
        try {
            FileInputStream fstream = new FileInputStream(path);
            br =
                new BufferedReader(new InputStreamReader(new DataInputStream(
                    fstream)));
            String strLine;
            StringBuffer buf = new StringBuffer();
            while ((strLine = br.readLine()) != null) {
                if (!estaVacia(strLine)) {
                    buf.append(strLine);
                    buf.append("\n");
                }
            }
            String result = buf.toString();
            if (result.length() != 0) {
                result = result.substring(0, result.length() - 1);
            }
            out = new PrintWriter(path);
            out.print(result);
        } catch (IOException e) {
```

```
            LOG.log(Level.SEVERE, "ERROR. Problem accessing \"" + path
                + "\" file");
        } finally {
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
            if (out != null) {
                out.close();
            }
        }
    }

    /**
     * Metodo que indica si una cadena contiene otros caracteres que espacios y
     * tabuladores.
     *
     * @param linea Cadena a analizar (se asume que es una linea, y por tanto
     *        que no contiene saltos de linea)
     * @return true si la cadena no contiene caracteres que no sean espacios y
     *         tabuladores, false en caso contrario
     */
    private static boolean estaVacia(String linea) {

        for (int i = 0; i < linea.length(); i++) {
            if (linea.charAt(i) != ' ' && linea.charAt(i) != '\t') {
                return false;
            }
        }
        return true;
    }

    /**
     * Copia todo el contenido de un directorio a otro directorio. Si este
     * ultimo no existe, lo crea.
     *
```

```
     * @param dirOrigen Ruta completa del directorio a copiar
     * @param dirDestino Ruta completa del directorio al que se copiara el
     *        contenido del anterior
     */
    public static void copyDirContent(String dirOrigen, String dirDestino) {

        try {
            File origen = new File(dirOrigen);
            File destino = new File(dirDestino);
            if (!origen.exists() || !origen.isDirectory()) {
                throw new IOException();
            }
            if (!destino.exists() && !destino.mkdir()) {
                problemCreatingDir(destino.getAbsolutePath());
            }
            File[] cont = origen.listFiles();
            for (int i = 0; i < cont.length; i++) {
                if (cont[i].isDirectory()) {
                    FileUtils.copyDirectory(cont[i], new File(dirDestino
                        + BARRA + cont[i].getName()));
                } else {
                    FileUtils.copyFile(cont[i], new File(dirDestino + BARRA
                        + cont[i].getName()));
                }
            }
        } catch (IOException ex) {
            LOG.log(Level.SEVERE, "ERROR. Cannot copy \"" + dirOrigen
                + "\" directory");
        }
    }

    /**
     * Obtiene el contenido de un fichero de texto contenido en el directorio
     * "resources" del JAR.
     *
     * @param internalFilePath Ruta del fichero de texto dentro de la raiz de
     *        "resources"
     * @param internalPath Ruta de la raiz de "resources", donde se encuentran
     *        realmente todos los recursos alli almacenados
```

```
     * @param cl Objeto ClassLoader necesario para acceder a los "resources" del
     *        JAR
     * @return Contenido del archivo de texto perteneciente a los "resources"
     *         del JAR indicado
     */
    public static String readTextFile(String internalFilePath,
                                      String internalPath, ClassLoader cl) {

        StringBuffer fileData = new StringBuffer(MIL);
        BufferedReader reader = null;
        try {
            System.setProperty(FILE_ENCODING, UTF8);
            InputStream is =
                cl.getResourceAsStream(internalPath + internalFilePath);
            reader = new BufferedReader(new InputStreamReader(is, UTF8));
            char[] buf = new char[BIN_KILO];
            int numRead = 0;
            while ((numRead = reader.read(buf)) != -1) {
                String readData = String.valueOf(buf, 0, numRead);
                fileData.append(readData);
                buf = new char[BIN_KILO];
            }
            return fileData.toString();
        } catch (IOException ex) {
            return "";
        } finally {
            try {
                if (reader != null) {
                    reader.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Obtiene la primera linea de un archivo de configuracion.
     *
```

```
     * @param confFilePath Ruta interna del archivo, dentro del directorio de
     *        configuracion
     * @return La primera linea del archivo
     * @throws IOException Si el archivo no existe o no tiene contenido
     */
    public static String getFirstLineConfFile(String confFilePath)
        throws IOException {

        BufferedReader br = null;
        try {
            br =
                new BufferedReader(new FileReader(
                    FileUtilities.getConfigurationFilePath(confFilePath)));
            String linea = br.readLine();
            br.close();
            if (linea == null) {
                throw new IOException();
            }
            return linea;
        } finally {
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Informa al usuario de que se ha producido un error al tratar de cerrar un
     * fichero.
     */
    public static void problemClosingFile() {

        LOG.log(Level.SEVERE, "ERROR. Problem closing file");
    }
```

```
    /**
     * Informa al usuario de que se ha producido un error al tratar de crear un
     * directorio.
     *
     * @param dir Nombre del directorio cuya creacion ha fallado
     */
    public static void problemCreatingDir(String dir) {

        LOG.log(Level.SEVERE, "ERROR. Problem creating \"" + dir
            + "\" directory.");
    }
}
```

PomUtils.java

```
package rooplusplus.roo.addon.utils;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.apache.commons.lang3.text.StrSubstitutor;
import org.springframework.roo.process.manager.FileManager;
import org.springframework.roo.project.ProjectOperations;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

/**
 * Utilidades para la edicion de ficheros pom.xml de Maven.
```

```
 *
 * @author javier.j.menendez
 */
public final class PomUtils {

    /**
     * Constructor privado para que esta clase de utilidades no pueda ser
     * instanciada.
     */
    private PomUtils() {

    }

    /**
     * Logger para informar al usuario de diferentes eventos.
     */
    private static final Logger LOG = Logger.getLogger("PomUtils.class");

    /**
     * Cadena con el caracter "/", usado en varias ocasiones en la clase.
     */
    private static final String BARRA = "/";

    /**
     * Cadena con el caracter "']", usado en varias ocasiones en la clase.
     */
    private static final String COMILLA_CORCHETE = "']";

    /**
     * Cadena con el caracter "-", usado en varias ocasiones en la clase.
     */
    private static final String GUION = "-";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String ARTIFACT_ID = "project.artifactId";

    /**
```

```
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String ERROR_1 = "ERROR. \"";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String ERROR_2 =
        "\" file not found or has wrong content";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String SCOPE = "scope";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String PROJECT_PROPERTIES = "/project/properties";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String PROJECT = "/project";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String PROJECT_BUILD = "/project/build";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String PROJECT_BUILD_PLUGINS =
        "/project/build/plugins";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
```

```
private static final String PLATFORM_VERSION = "${platform.version}";

/**
 * Asigna los valores correspondientes a las diferentes variables en un
 * fichero pom.xml: nombre, groupId, artifactId y version del proyecto y de
 * su parent.
 *
 * @param pomPath Ruta completa del fichero pom.xml en el que se asignaran
 *        los valores a las variables
 * @param moduleName Nombre del modulo al que pertenece dicho pom.xml
 * @param projectName Nombre del proyecto al que pertenece dicho pom.xml
 * @param version Version del proyecto, y por tanto de sus modulos
 * @param topLevelPackage Paquete Java correspondiente al modulo (es decir,
 *        groupId)
 */
public static void replaceVariables(String pomPath, String moduleName,
                                     String projectName, String version,
                                     String topLevelPackage) {

    Map<String, String> valuesMap =
        createValuesMap(projectName, version, topLevelPackage, moduleName);
    String templateString, resolvedString;
    StrSubstitutor sub;
    templateString = FileUtilities.readFileAsString(pomPath);
    sub = new StrSubstitutor(valuesMap);
    resolvedString = sub.replace(templateString);
    BufferedWriter out = null;
    try {
        out = new BufferedWriter(new FileWriter(pomPath));
        out.write(resolvedString);
    } catch (IOException ex) {
        fileNotFoundException(pomPath);
    } finally {
        try {
            if (out != null) {
                out.close();
            }
        } catch (IOException ex) {
            FileUtilities.problemClosingFile();
```

```
        }
    }
}

/**
 * Crea un Map de Strings con los pares de valores "sustituto/sustituido"
 * para luego Paquetelos en un fichero pom.xml, configurando de esta
 * forma un modulo de un proyecto de la manera adecuada.
 *
 * @param projectName Nombre del proyecto al que pertenece dicho pom.xml
 * @param version Version del proyecto, y por tanto de sus modulos
 * @param topLevelPackage Paquete Java correspondiente al modulo (es decir,
 *        groupId)
 * @param moduleName Nombre del modulo al que pertenece dicho pom.xml
 * @return Map de Strings con los pares de valores "sustituto/sustituido" a
 *         Paquete el el pom.xml para configurar el modulo del proyecto
 */
private static Map<String, String> createValuesMap(String projectName,
                                                    String version,
                                                    String topLevelPackage,
                                                    Object moduleName) {

    Map<String, String> valuesMap = new HashMap<String, String>();
    valuesMap.put("project", projectName);
    valuesMap.put("project.name", projectName);
    valuesMap.put("project.parent.name", projectName);
    valuesMap.put("project.version", version);
    valuesMap.put("project.parent.version", version);
    valuesMap.put("project.parent.artifactId", projectName);
    valuesMap.put("project.groupId", topLevelPackage);
    valuesMap.put("project.parent.groupId", topLevelPackage);
    if (projectName.equals(moduleName)) {
        valuesMap.put(ARTIFACT_ID, projectName);
    } else {
        valuesMap.put(ARTIFACT_ID, projectName + GUION + moduleName);
    }
    return valuesMap;
}
```

```
/**
 * Introduce como hija del nodo "dependencies", hijo de la raiz del pom.xml
 * ("project"), una nueva dependencia.
 *
 * @param pomPath Ruta completa del fichero pom.xml en el que se creara la
 *        nueva dependencia
 * @param groupId groupId de la nueva dependencia
 * @param artifactId artifactId de la nueva dependencia
 * @param version version de la nueva dependencia
 * @throws TransformerException Si el pom.xml indicado no tiene el contenido
 *         esperado
 * @throws IOException Si el pom.xml indicado no existe
 * @throws SAXException Si el pom.xml indicadono tiene el nodo
 *         "dependencies" hijo de la raiz, "project"
 * @throws ParserConfigurationException Si el pom.xml indicadono tiene el
 *         nodo "dependencies" hijo de la raiz, "project"
 * @throws XPathExpressionException Si el pom.xml indicadono tiene el nodo
 *         "dependencies" hijo de la raiz, "project"
 */
public static void addDependency(String pomPath, String groupId,
                                 String artifactId, String version)
    throws XPathExpressionException,
        ParserConfigurationException,
        SAXException,
        IOException,
        TransformerException {

    addDependency("/project/dependencies", pomPath, groupId, artifactId,
        version);
}

/**
 * Introduce una nueva dependencia en uno de los "build/plugins" de un
 * fichero pom.xml.
 *
 * @param pomPath Ruta completa del fichero pom.xml en el que se introducira
 *        la dependencia del plugin
 * @param pluginArtifactId artifactId del plugin al que se pondra la
 *        dependencia
```

```
     * @param groupId groupId de la dependencia a introducir
     * @param artifactId artifactId de la dependencia a introducir
     * @param version version de la dependencia a introducir
     * @throws TransformerException Si el fichero pom.xml indicado no tiene el
     *         contenido adecuado, o si no existe el plugin indicado dentro del
     *         mismo
     * @throws IOException Si el fichero pom.xml indicado no existe
     * @throws SAXException Si el fichero pom.xml indicado no tiene el contenido
     *         adecuado, o si no existe el plugin indicado dentro del mismo
     * @throws ParserConfigurationException Si el fichero pom.xml indicado no
     *         tiene el contenido adecuado, o si no existe el plugin indicado
     *         dentro del mismo
     * @throws XPathExpressionException Si no existe el plugin indicado dentro
     *         del fichero pom.xml indicado
     */
    public static void addDependencyToBuildPlugin(String pomPath,
                                                   String pluginArtifactId,
                                                   String groupId,
                                                   String artifactId,
                                                   String version)
        throws XPathExpressionException,
            ParserConfigurationException,
            SAXException,
            IOException,
            TransformerException {

        addDependency("/project/build/plugins/plugin/artifactId[text()='"
            + pluginArtifactId + "']/../dependencies", pomPath, groupId,
            artifactId, version);
    }

    /**
     * Introduce una nueva dependencia en el lugar indicado de un fichero
     * pom.xml.
     *
     * @param xPath Ubicacion (nodo padre) en la que se ha de introducir la
     *        nueva dependencia
     * @param pomPath Ruta completa del fichero pom.xml en el que se introducira
     *        la dependencia
```

```
     * @param groupId groupId de la nueva dependencia
     * @param artifactId artifactId de la nueva dependencia
     * @param version version de la nueva dependencia
     * @throws IOException Si no existe el pom.xml indicado
     * @throws SAXException Si el pom.xml indicado no tiene el contenido
     *         adecuado, o si la ubicacion en la que se creara la dependencia no
     *         es valida o no existe
     * @throws ParserConfigurationException Si el pom.xml indicado no tiene el
     *         contenido adecuado, o si la ubicacion en la que se creara la
     *         dependencia no es valida o no existe
     * @throws XPathExpressionException Si el pom.xml indicado no tiene el
     *         contenido adecuado, o si la ubicacion en la que se creara la
     *         dependencia no es valida o no existe
     * @throws TransformerException Si el pom.xml indicado no tiene el contenido
     *         adecuado, o si la ubicacion en la que se creara la dependencia no
     *         es valida o no existe
     */
    private static void addDependency(String xPath, String pomPath,
                                      String groupId, String artifactId,
                                      String version)
        throws ParserConfigurationException,
            SAXException,
            IOException,
            XPathExpressionException,
            TransformerException {

        Document doc = XmlUtils.createDocumentFromFile(pomPath);
        // Si ya existe la dependencia a introducir salimos sin hacer nada
        String existe =
            xPath + "/dependency/artifactId[text()='" + artifactId
                + COMILLA_CORCHETE;
        NodeList siExiste = XmlUtils.evaluateXPath(existe, doc);
        if (siExiste.getLength() > 0) {
            return;
        }
        Element gId = doc.createElement("groupId");
        gId.setTextContent(groupId);
        Element aId = doc.createElement("artifactId");
        aId.setTextContent(artifactId);
```

```
        Element ver = doc.createElement("version");
        ver.setTextContent(version);
        Element dep = doc.createElement("dependency");
        dep.appendChild(gId);
        dep.appendChild(aId);
        dep.appendChild(ver);
        NodeList deps = XmlUtils.evaluateXPath(xPath, doc);
        if (deps.getLength() > 0) {
            deps.item(0).appendChild(dep);
        }
        // Si no existe la seccion dependencies la creamos
        else {
            doc = createDependenciesElement(xPath, dep, doc);
        }
        // Guardar los resultados
        XmlUtils.guardaXml(doc, pomPath);
    }

    /**
     * Introduce el elemento "dependencies" en el lugar indicado en un documento
     * XML correspondiente a un fichero pom.xml, introduciendo como hijo del
     * mismo el elemento indicado como parametro.
     *
     * @param xPath XPath completo del nodo "dependencies" a crear
     * @param dep Dependencia que sera creada al mismo tiempo que el nodo
     *        "dependencies", como hija del mismo
     * @param doc Documento XML en el que se introducira la seccion
     *        "dependencies"
     * @return El documento XML con la seccion "dependencies" y una primera
     *         dependencia creadas
     * @throws XPathExpressionException Si el XPath que indica el lugar en el
     *         que se ha de introducir la nueva seccion "dependencies" no es
     *         valido para ese documento, o si la primera dependencia a
     *         introducir procede de otro documento
     */
    private static Document createDependenciesElement(String xPath,
                                                       Element dep, Document doc)
        throws XPathExpressionException {
```

```
        String[] xpathsplit = xPath.split(BARRA);
        StringBuffer buf = new StringBuffer();
        for (int i = 1; i < xpathsplit.length - 1; i++) {
            buf.append(BARRA);
            buf.append(xpathsplit[i]);
        }
        String xpath = buf.toString();
        Node proj = XmlUtils.evaluateXPath(xpath, doc).item(0);
        Element dependencies = doc.createElement("dependencies");
        dependencies.appendChild(dep);
        proj.appendChild(dependencies);
        return doc;
    }

    /**
     * Introduce un elemento SCOPE como hijo del nodo indicado por el XPath de
     * su artifactId hijo.
     *
     * @param value Valor (contenido) del nuevo nodo XML SCOPE
     * @param artifactIdxPath XPath del artifactId del elemento al que se
     *        introducira el nodo SCOPE como hijo
     * @param doc Documento XML en el que se introducira el nuevo nodo
     * @param pomPath Path del fichero pom.xml en el que se introducira el nuevo
     *        nodo
     * @throws XPathExpressionException Si el XPath del artifactId del elemento
     *         no corresponde a un nodo que realmente exista en el documento
     * @throws TransformerException Si el fichero pom.xml indicado no existe o
     *         no tiene un formato correcto
     */
    public static void addScope(String value, String artifactIdxPath,
                                Document doc, String pomPath)
        throws XPathExpressionException,
            TransformerException {

        Element scope = doc.createElement(SCOPE);
        scope.setTextContent(value);
        NodeList list = XmlUtils.evaluateXPath(artifactIdxPath, doc);
        for (int i = 0; i < list.getLength(); i++) {
            Node parent = list.item(i).getParentNode();
```

```
            parent.appendChild(scope);
        }
        // Guardar los resultados
        XmlUtils.guardaXml(doc, pomPath);
    }

    /**
     * Introduce una nueva propiedad en el campo "properties" del fichero
     * pom.xml indicado.
     *
     * @param pomPath Ruta completa del fichero pom.xml en el que se introducira
     *        la propiedad
     * @param name Nombre de la propiedad a introducir
     * @param value Valor de la propiedad a introducir
     * @throws IOException Si el pom.xml indicado no existe o no tiene un
     *         formato valido
     * @throws SAXException Si el pom.xml no contiene los nodos "project" y
     *         "properties"
     * @throws ParserConfigurationException Si el pom.xml no contiene los nodos
     *         "project" y "properties"
     * @throws XPathExpressionException Si el pom.xml no contiene los nodos
     *         "project" y "properties"
     */
    public static void addProperty(String pomPath, String name, String value)
        throws XPathExpressionException,
            ParserConfigurationException,
            SAXException,
            IOException {

        NodeList propsList =
            XmlUtils.evaluateXPath(PROJECT_PROPERTIES,
                XmlUtils.createDocumentFromFile(pomPath));
        if (propsList.getLength() == 0) {
            XmlUtils.addChildToNode(pomPath, PROJECT, "properties", "");
        }
        XmlUtils.addChildToNode(pomPath, PROJECT_PROPERTIES, name, value);
    }

    /**
```

```
     * Establece en el pom.xml indicado por el parametro pomPath completo) la
     * version del proyecto, y por tanto, del modulo.
     *
     * @param pomPath Ruta completa del fichero pom.xml en el que se establecera
     *        la version
     * @param version Nombre (normalmente, con numero/s) de la version a
     *        establecer
     * @param fileManager Objeto FileManager necesario para asegurar que las
     *        operaciones de entrada/salida realmente han sido escritas en disco
     */
    public static void setVersion(String pomPath, String version,
                                  FileManager fileManager) {

        try {
            Document doc = XmlUtils.createDocumentFromFile(pomPath);
            // Sustituir '0.1.0.BUILD-SNAPSHOT' por la version que corresponda
            NodeList nodes =
                XmlUtils.evaluateXPath(
                    "/project/version[text()='0.1.0.BUILD-SNAPSHOT']", doc);
            for (int idx = 0; idx < nodes.getLength(); idx++) {
                nodes.item(idx).setTextContent(version);
            }
            // Guardar los resultados
            XmlUtils.guardaXml(doc, pomPath);
        } catch (IOException ex) {
            fileNotFoundException(pomPath);
        } catch (ParserConfigurationException e) {
            fileNotFoundException(pomPath);
        } catch (SAXException e) {
            fileNotFoundException(pomPath);
        } catch (XPathExpressionException e) {
            fileNotFoundException(pomPath);
        } catch (TransformerException e) {
            fileNotFoundException(pomPath);
        }
    }

    /**
     * Introduce en el pom.xml indicado el plugin de la licencia contenido en el
```

```
     * fichero plugin-license.xml del directorio de configuracion.
     *
     * @param pomPath Ruta completa del pom.xml en el que se introducira el
     *        plugin de la licencia
     */
    public static void addLicensePlugin(String pomPath) {

        try {
            // Leer plugin-license.xml e introducir su contenido en el pom.xml
            Document docPluginLicense =
                XmlUtils.createDocumentFromFile(FileUtilities
                    .getConfigurationFilePath("plugin-license.xml"));
            Node pluginLicenseRoot =
                XmlUtils.evaluateXPath("/plugin", docPluginLicense).item(0);
            addBuildPlugin(pomPath, pluginLicenseRoot);
        } catch (IOException ex) {
            fileNotFoundException(pomPath);
        } catch (SAXException e) {
            fileNotFoundException(pomPath);
        } catch (ParserConfigurationException e) {
            fileNotFoundException(pomPath);
        } catch (XPathExpressionException e) {
            fileNotFoundException(pomPath);
        }
    }

    /**
     * Introduce en el pom.xml indicado la licencia segun el contenido del
     * fichero licenses.xml situado en el directorio de configuracion.
     *
     * @param pomPath Ruta completa del pom.xml en el que se introducira la
     *        licencia
     */
    public static void addLicense(String pomPath) {

        try {
            // Leer licenses.xml e introducir su contenido en el pom.xml
            Document doc = XmlUtils.createDocumentFromFile(pomPath);
            // Si el pom ya tiene alguna licencia introducida salimos sin hacer
```

```
            // nada
            if (XmlUtils.evaluateXPath("/project/licenses", doc).getLength() !=
0) {
                return;
            }
            Node root = XmlUtils.evaluateXPath(PROJECT, doc).item(0);
            Document docLicenses =
                XmlUtils.createDocumentFromFile(FileUtilities
                    .getConfigurationFilePath("licenses.xml"));
            Node licensesRoot =
                XmlUtils.evaluateXPath("/licenses", docLicenses).item(0);
            Node licensesRootImported = doc.importNode(licensesRoot, true);
            root.appendChild(licensesRootImported);
            // Guardar los resultados
            XmlUtils.guardaXml(doc, pomPath);
        } catch (IOException ex) {
            fileNotFoundException(pomPath);
        } catch (ParserConfigurationException e) {
            fileNotFoundException(pomPath);
        } catch (SAXException e) {
            fileNotFoundException(pomPath);
        } catch (XPathExpressionException e) {
            fileNotFoundException(pomPath);
        } catch (TransformerException e) {
            fileNotFoundException(pomPath);
        }
    }

    /**
     * Introduce en el fichero pom.xml indicado, en project/build/plugins, el
     * nodo (conteniendo un plugin) indicado como parametro Si no existieran los
     * elementos build y/o plugins en el pom.xml, son creados.
     *
     * @param pomPath Ruta completa del pom.xml en el que se introducira el
     *        plugin
     * @param pluginAImportar Nodo XML que contiene el plugin a importar
     */
    public static void addBuildPlugin(String pomPath, Node pluginAImportar) {

        try {
```

```
            Document doc = XmlUtils.createDocumentFromFile(pomPath);
            Node plugin = doc.adoptNode(pluginAImportar.cloneNode(true));
            Node root = XmlUtils.evaluateXPath(PROJECT, doc).item(0);
            // Si no existe build en el pom
            if (XmlUtils.evaluateXPath(PROJECT_BUILD, doc).getLength() == 0) {
                Node build = doc.createElement("build");
                root.appendChild(build);
            }
            Node build = XmlUtils.evaluateXPath(PROJECT_BUILD, doc).item(0);
            // Si existe build en el pom pero no contiene plugins
            if (XmlUtils.evaluateXPath(PROJECT_BUILD_PLUGINS, doc).getLength()
== 0) {
                Node plugins = doc.createElement("plugins");
                build.appendChild(plugins);
            }
            // Si ya existe build y contiene plugins en el pom
            Node plugins =
                XmlUtils.evaluateXPath(PROJECT_BUILD_PLUGINS, doc).item(0);
            plugins.appendChild(plugin);
            // Guardar los resultados
            XmlUtils.guardaXml(doc, pomPath);
        } catch (IOException ex) {
            fileNotFoundException(pomPath);
        } catch (ParserConfigurationException e) {
            fileNotFoundException(pomPath);
        } catch (SAXException e) {
            fileNotFoundException(pomPath);
        } catch (XPathExpressionException e) {
            fileNotFoundException(pomPath);
        } catch (TransformerException e) {
            fileNotFoundException(pomPath);
        }
    }

    /**
     * Introduce en un pom la dependencia con respecto al modulo del mismo
     * nombre de la plataforma a utilizar, y al modulo de test de dicha
     * plataforma que siempre es utilizado.
     *
     * @param module Nombre del modulo al que se le pondra la dependencia
```

```
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, que permite conocer que modulo y proyecto tienen el
     *        foco
     * @param fileManager Objeto FileManager necesario para poder garantizar que
     *        los cambios sobre los ficheros realmente se han escrito en disco
     * @param platformName Nombre de la plataforma
     * @param groupId groupId de la plataforma
     * @param testModule Nombre del modulo de test de la plataforma
     * @throws TransformerException Si el pom.xml del modulo indicado no existe
     *         en el lugar esperado o no tiene el contenido adecuado
     * @throws IOException Si el pom.xml del modulo indicado no existe en el
     *         lugar esperado o no tiene el contenido adecuado
     * @throws SAXException Si el pom.xml del modulo indicado no existe en el
     *         lugar esperado o no tiene el contenido adecuado
     * @throws ParserConfigurationException Si el pom.xml del modulo indicado no
     *         existe en el lugar esperado o no tiene el contenido adecuado
     * @throws XPathExpressionException Si el pom.xml del modulo indicado no
     *         existe en el lugar esperado o no tiene el contenido adecuado
     */
    public static void addPlatformDependency(String module,
                                             ProjectOperations
projectOperations,
                                             FileManager fileManager,
                                             String platformName,
                                             String groupId, String testModule)
        throws XPathExpressionException,
            ParserConfigurationException,
            SAXException,
            IOException,
            TransformerException {

        String pomPath =
            ProjectUtils.getProjectPath(projectOperations) + BARRA
                + projectOperations.getFocusedProjectName() + GUION + module
                + "/pom.xml";
        addDependency(pomPath, groupId, platformName + GUION + module,
            PLATFORM_VERSION);
        addDependency(pomPath, groupId, platformName + GUION + testModule,
            PLATFORM_VERSION);
        // Indicar <scope>test</scope> en la dependencia respecto al modulo de
```

```
        // test
        String xPath =
            "/project/dependencies/dependency/artifactId[text()='"
                + platformName + GUION + testModule + COMILLA_CORCHETE;
        Document doc = XmlUtils.createDocumentFromFile(pomPath);
        Node parent =
            XmlUtils.evaluateXPath(xPath, doc).item(0).getParentNode();
        Element node = doc.createElement(SCOPE);
        node.setTextContent("test");
        parent.appendChild(node);
        XmlUtils.guardaXml(doc, pomPath);
    }

    /**
     * Informa al usuario cuando se produce una excepcion porque un fichero
     * pom.xml no existe o no tiene el contenido adecuado.
     *
     * @param pomPath Path del fichero XML incorrecto
     */
    private static void fileNotFoundException(String pomPath) {

        LOG.log(Level.SEVERE, ERROR_1 + pomPath + ERROR_2);
    }
}
```

ProjectUtils.java

```
package rooplusplus.roo.addon.utils;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
```

```
import java.io.PrintWriter;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.springframework.roo.project.Path;
import org.springframework.roo.project.PathResolver;
import org.springframework.roo.project.ProjectOperations;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

/**
 * Utilidades para la gestion de proyectos Maven/Spring Roo.
 *
 * @author javier.j.menendez
 */
public final class ProjectUtils {

    /**
     * Constructor privado para que esta clase de utilidades no pueda ser
     * instanciada.
     */
    private ProjectUtils() {

    }

    /**
     * Log para mostrar informacion al usuario.
     */
    private static final Logger LOG = Logger.getLogger("ProjectUtils.class");

    /**
     * Cadena con el caracter "/", usado en varias ocasiones en la clase.
     */
    private static final String BARRA = "/";
```

```
/**
 * Cadena con el caracter "-", usado en varias ocasiones en la clase.
 */
private static final String GUION = "-";

/**
 * Cadena con el caracter "\n", usado en varias ocasiones en la clase.
 */
private static final String LINEA = "\n";

/**
 * Cadena con el caracter "\"", usado en varias ocasiones en la clase.
 */
private static final String COMILLAS = "\"";

/**
 * Cadena usada en varias ocasiones en la clase.
 */
private static final String ERROR_CREATING_FILE =
    "ERROR. Problem creating file \"";

/**
 * Cadena usada en varias ocasiones en la clase.
 */
private static final String FILE_NOT_FOUND_1 = "ERROR. File \"";

/**
 * Cadena usada en varias ocasiones en la clase.
 */
private static final String FILE_NOT_FOUND_2 = "\" not found";

/**
 * Cadena usada en varias ocasiones en la clase.
 */
private static final String FILE_ENCODING = "file.encoding";

/**
 * Cadena usada en varias ocasiones en la clase.
```

```
     */
    private static final String UTF8 = "UTF-8";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String DEPENDENCIES = "dependencies";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String POM = "pom.xml";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String WEBXML_ERROR =
        "ERROR. \"web.xml\" file not found or has wrong content";

    /**
     * Cadena "postgresql", usada en varias ocasiones en la clase.
     */
    private static final String POSTGRESQL = "postgresql";

    /**
     * Cadena "ojdbc", usada en varias ocasiones en la clase.
     */
    private static final String OJDBC = "ojdbc";

    /**
     * Cadena "jtds", usada en varias ocasiones en la clase.
     */
    private static final String JTDS = "jtds";

    /**
     * Cadena "mysql", usada en varias ocasiones en la clase.
     */
    private static final String MYSQL = "mysql";
```

```
/**
 * Copia un fichero binario desde el directorio "resources" del JAR hasta el
 * proyecto.
 *
 * @param projectPath Ruta completa del directorio raiz del proyecto
 * @param projectName Nombre del proyecto
 * @param 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 ubicacion a la que sera copiado dentro del
 *        proyecto, excepto por el nombre del directorio del modulo que en
 *        el proyecto incluira el nombre de este ultimo)
 * @param internalPath Ruta de la raiz del directorio de resources del JAR
 * @param cl Objeto ClassLoader necesario para obtener el fichero contenido
 *        en "resources" en el JAR
 */
public static void copyBinaryFile(String projectPath, String projectName,
                                  String internalFilePath,
                                  String internalPath, ClassLoader cl) {

    InputStream is = null;
    FileOutputStream os = null;
    try {
        is = cl.getResourceAsStream(internalPath + internalFilePath);
        File file;
        if (internalFilePath.contains(BARRA)) {
            file =
                new File(projectPath + BARRA + projectName + GUION
                    + internalFilePath);
        } else {
            file = new File(projectPath + BARRA + internalFilePath);
        }
        os = new FileOutputStream(file);
        int val = 0;
        do {
            val = is.read();
            os.write(val);
        } while (val != -1);
    } catch (IOException ex) {
```

```
            LOG.log(Level.SEVERE, ERROR_CREATING_FILE + internalFilePath
                + COMILLAS);
        } finally {
            try {
                if (is != null) {
                    is.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
            try {
                if (os != null) {
                    os.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Copia un fichero binario al proyecto.
     *
     * @param projectPath Ruta completa del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     * @param internalFilePath Ruta destino del fichero a copiar, partiendo del
     *        directorio del modulo que lo contiene (por ejemplo,
     *        "web/fichero.pdf", sin incluir en ella el nombre del proyecto)
     * @param filePath Ruta completa del fichero a copiar
     */
    public static void copyExternBinaryFile(String projectPath,
                                            String projectName,
                                            String internalFilePath,
                                            String filePath) {

        InputStreamReader is = null;
        FileOutputStream os = null;
        try {
            is = new InputStreamReader(new FileInputStream(filePath));
```

```
            File file;
            if (internalFilePath.contains(BARRA)) {
                file =
                    new File(projectPath + BARRA + projectName + GUION
                        + internalFilePath);
            } else {
                file = new File(projectPath + BARRA + internalFilePath);
            }
            os = new FileOutputStream(file);
            int val = 0;
            do {
                val = is.read();
                os.write(val);
            } while (val != -1);
        } catch (IOException ex) {
            LOG.log(Level.SEVERE, FILE_NOT_FOUND_1 + filePath
                + FILE_NOT_FOUND_2);
        } finally {
            try {
                if (is != null) {
                    is.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
            try {
                if (os != null) {
                    os.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Copia un fichero de texto desde el directorio "resources" del JAR hasta
     * el proyecto.
     *
```

```
     * @param projectPath Ruta completa del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     * @param 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 ubicacion a la que sera copiado dentro del
     *        proyecto, excepto por el nombre del directorio del modulo que en
     *        el proyecto incluira el nombre de este ultimo)
     * @param internalPath Ruta de la raiz del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para obtener el fichero contenido
     *        en "resources" en el JAR
     */
    public static void copyTextFile(String projectPath, String projectName,
                                    String internalFilePath,
                                    String internalPath, ClassLoader cl) {

        BufferedReader in = null;
        OutputStreamWriter out = null;
        try {
            System.setProperty(FILE_ENCODING, UTF8);
            in =
                new BufferedReader(new InputStreamReader(
                    cl.getResourceAsStream(internalPath + internalFilePath),
                    UTF8));
            File file;
            if (internalFilePath.contains(BARRA)) {
                file =
                    new File(projectPath + BARRA + projectName + GUION
                        + internalFilePath);
            } else {
                file = new File(projectPath + BARRA + internalFilePath);
            }
            FileOutputStream fileOutputStream = new FileOutputStream(file);
            out = new OutputStreamWriter(fileOutputStream, UTF8);
            String str;
            while ((str = in.readLine()) != null) {
                if (in.ready()) {
                    str = str + LINEA;
                }
```

```
                out.write(str, 0, str.length());
            }
        } catch (IOException ex) {
            LOG.log(Level.SEVERE, ERROR_CREATING_FILE + internalFilePath
                + COMILLAS);
        } finally {
            try {
                if (in != null) {
                    in.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
            try {
                if (out != null) {
                    out.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Copia un fichero de texto al proyecto.
     *
     * @param projectPath Ruta completa del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     * @param internalFilePath Ruta destino del fichero a copiar, partiendo del
     *        directorio del modulo que lo contiene (por ejemplo,
     *        "web/fichero.txt", sin incluir en ella el nombre del proyecto)
     * @param filePath Ruta completa del fichero a copiar
     */
    public static void copyExternTextFile(String projectPath,
                                          String projectName,
                                          String internalFilePath,
                                          String filePath) {

        BufferedReader in = null;
```

```
OutputStreamWriter out = null;
try {
    System.setProperty(FILE_ENCODING, UTF8);
    in = new BufferedReader(new FileReader(filePath));
    File file;
    if (internalFilePath.contains(BARRA)) {
        file =
            new File(projectPath + BARRA + projectName + GUION
                + internalFilePath);
    } else {
        file = new File(projectPath + BARRA + internalFilePath);
    }
    FileOutputStream fileOutputStream = new FileOutputStream(file);
    out = new OutputStreamWriter(fileOutputStream, UTF8);
    String str;
    while ((str = in.readLine()) != null) {
        if (in.ready()) {
            str = str + LINEA;
        }
        out.write(str, 0, str.length());
    }
} catch (IOException ex) {
    LOG.log(Level.SEVERE, FILE_NOT_FOUND_1 + filePath
        + FILE_NOT_FOUND_2);
} finally {
    try {
        if (in != null) {
            in.close();
        }
    } catch (IOException ex) {
        FileUtilities.problemClosingFile();
    }
    try {
        if (out != null) {
            out.close();
        }
    } catch (IOException ex) {
        FileUtilities.problemClosingFile();
    }
```

```
    }
}

/**
 * Introduce un texto en la parte final de un fichero de texto preexistente,
 * dentro de un proyecto. Si el archivo no existe, es creado.
 *
 * @param text Texto a introducir
 * @param projectPath Ruta del directorio raiz del proyecto
 * @param projectName Nombre del proyecto
 * @param internalFilePath Ruta del fichero en el que se introducira el
 *        texto, partiendo del directorio del modulo que lo contiene (por
 *        ejemplo, "web/fichero.txt", sin incluir en ella el nombre del
 *        proyecto)
 */
public static void appendTextToFile(String text, String projectPath,
                                    String projectName,
                                    String internalFilePath) {

    PrintWriter out = null;
    try {
        System.setProperty(FILE_ENCODING, UTF8);
        File file;
        if (internalFilePath.contains(BARRA)) {
            file =
                new File(projectPath + BARRA + projectName + GUION
                    + internalFilePath);
        } else {
            file = new File(projectPath + BARRA + internalFilePath);
        }
        boolean file_existed = file.exists();
        if (!file_existed && !file.createNewFile()) {
            throw new IOException();
        }
        out =
            new PrintWriter(new BufferedWriter(new FileWriter(
                file.getAbsolutePath(), true)));
        if (file_existed) {
            text = LINEA + text;
```

```
            }
            out.print(text);
        } catch (IOException ex) {
            LOG.log(Level.SEVERE, "File " + internalFilePath
                + " did not exist and could not be created.");
        } finally {
            if (out != null) {
                out.close();
            }
        }
    }

    /**
     * Obtiene la ruta completa del directorio raiz del proyecto. Solo
     * funcionara de la forma esperada si el foco se encuentra en la raiz de
     * dicho proyecto, y no en uno de sus modulos.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @return Ruta completa del directorio raiz del proyecto
     */
    public static String getProjectPath(ProjectOperations projectOperations) {

        return (new File(projectOperations.getPathResolver()
            .getFocusedIdentifier(Path.ROOT, "."))).getPath();
    }

    /**
     * Determina si los diferentes modulos han sido creados en el proyecto (es
     * decir, si se ha ejecutado create-modules). No funcionara de la forma
     * esperada si se han introducido manualmente cambios inapropiados en el
     * pom.xml raiz del proyecto. Tampoco detectara si se han eliminado a mano
     * uno o varios de los modulos previamente creados.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @return true en caso de ya que se hayan creado los modulos del proyecto;
```

```
     *         false en caso contrario
     */
    public static boolean areModulesCreated(ProjectOperations projectOperations)
{

        PathResolver pathResolver = projectOperations.getPathResolver();
        File f = new File(pathResolver.getFocusedIdentifier(Path.ROOT, POM));
        if (!f.exists()) {
            return false;
        }
        String pom = FileUtilities.readFileAsString(f.getAbsolutePath());
        return pom.contains("-ear") && pom.contains("-package")
            && pom.contains("-web");
    }

    /**
     * Configura los plugins maven de tomcat y jetty en el fichero pom.xml del
     * modulo web, de forma que utilicen la base de datos indicada en el mismo
     * fichero, caso de que haya alguna.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    public static void setMavenPlugins(ProjectOperations projectOperations) {

        // Obtenemos el pom.xml del modulo web
        File f = WebModuleUtils.getWebModuleFile(POM, projectOperations);
        String pom = FileUtilities.readFileAsString(f.getAbsolutePath());
        // Invocamos a WebModuleUtils.setMavenPlugins de acuerdo al SGBD que
        // aparezca en el
        // pom.xml del modulo web, o con la cadena vacia si no aparece ninguno
        if (pom.contains(OJDBC)) {
            WebModuleUtils.setMavenPlugins(OJDBC, f.getAbsolutePath());
        } else if (pom.contains(POSTGRESQL)) {
            WebModuleUtils.setMavenPlugins(POSTGRESQL, f.getAbsolutePath());
        } else if (pom.contains(MYSQL)) {
            WebModuleUtils.setMavenPlugins(MYSQL, f.getAbsolutePath());
        } else if (pom.contains(JTDS)) {
            WebModuleUtils.setMavenPlugins(JTDS, f.getAbsolutePath());
```

```
        } else {
            WebModuleUtils.setMavenPlugins("", f.getAbsolutePath());
        }
    }

    /**
     * Elimina las dependencias de los plugins maven de tomcat y jetty en el
     * fichero pom.xml del modulo web del proyecto, de forma que pueda volver a
     * realizarse un "web deploy" sin que haya elementos repetidos u otros
     * problemas relacionados.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    public static void undoWebDeploy(ProjectOperations projectOperations) {

        // Obtenemos el pom.xml del modulo web
        File f = WebModuleUtils.getWebModuleFile(POM, projectOperations);
        try {
            // Deshacer las partes del web deploy anterior que puedan traer
            // problemas
            Document doc = XmlUtils.createDocumentFromFile(f.getAbsolutePath());
            String xPath =
                "/project/build/plugins/plugin/artifactId[text()='tomcat-maven-
plugin']";
            NodeList list =
                XmlUtils.evaluateXPath(xPath, doc).item(0).getParentNode()
                    .getChildNodes();
            for (int i = 0; i < list.getLength(); i++) {
                if ("configuration".equals(list.item(i).getNodeName())
                    || DEPENDENCIES.equals(list.item(i).getNodeName())) {
                    list.item(i).getParentNode().removeChild(list.item(i));
                }
            }
            xPath =
                "/project/build/plugins/plugin/artifactId[text()='maven-jetty-
plugin']";
            list =
                XmlUtils.evaluateXPath(xPath, doc).item(0).getParentNode()
```

```
                    .getChildNodes();
            for (int i = 0; i < list.getLength(); i++) {
                if (DEPENDENCIES.equals(list.item(i).getNodeName())) {
                    list.item(i).getParentNode().removeChild(list.item(i));
                }
            }
            XmlUtils.guardaXml(doc, f.getAbsolutePath());
        } catch (IOException ex) {
            webPomNotFound();
        } catch (ParserConfigurationException e) {
            webPomNotFound();
        } catch (SAXException e) {
            webPomNotFound();
        } catch (XPathExpressionException e) {
            webPomNotFound();
        } catch (TransformerException e) {
            webPomNotFound();
        }
    }

    /**
     * Informa al usuario de que hubo un problema con el fichero pom.xml del
     * modulo web.
     */
    private static void webPomNotFound() {

        LOG.log(Level.SEVERE,
            "ERROR. Web module's \"pom.xml\" file not found or has wrong
content");
    }

    /**
     * Introduce en el fichero "src/main/webapp/WEB-INF/web.xml", dentro del
     * modulo "web" del proyecto, la referencia al DataSource correspondiente,
     * segun el nombre del proyecto, para que Jetty funcione correctamente.
     *
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param internalPath Ruta de la raiz de los resources dentro del JAR
     * @param projectOperations Objeto ProjectOperations necesario para conocer
     *        la ubicacion de los archivos del proyecto.
```

```
     * @param cl Objeto ClassLoader necesario para obtener los archivos
     *        contenidos en el directorio "resources" del JAR
     */
    public static void dataSourceWebXml(String projectPath,
                                        String internalPath,
                                        ProjectOperations projectOperations,
                                        ClassLoader cl) {

        String resourceRef =
            FileUtilities
                .readTextFile("web/resource-ref.xml", internalPath, cl);
        resourceRef =
            resourceRef.replace("testDS",
                projectOperations.getFocusedProjectName() + "DS");
        String docPath =
            projectPath + BARRA + projectOperations.getFocusedProjectName()
                + "-web/src/main/webapp/WEB-INF/web.xml";
        try {
            Document add = XmlUtils.createDocumentFromString(resourceRef);
            Document doc = XmlUtils.createDocumentFromFile(docPath);
            doc.getDocumentElement().appendChild(
                doc.adoptNode(add.getDocumentElement().cloneNode(true)));
            XmlUtils.guardaXml(doc, docPath);
        } catch (IOException ex) {
            webxmlError();
        } catch (SAXException ex) {
            webxmlError();
        } catch (ParserConfigurationException ex) {
            webxmlError();
        } catch (TransformerException ex) {
            webxmlError();
        }
    }

    /**
     * Tratamiento de excepciones producidas durante la ejecucion del metodo
     * dataSourceWebXml.
     */
    private static void webxmlError() {
```

```
        LOG.log(Level.SEVERE, WEBXML_ERROR);
    }
}
```

UsermgmtUtils.java

```
package rooplusplus.roo.addon.utils;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.springframework.roo.project.ProjectOperations;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

/**
 * Utilidades para ayudar a introducir la gestion de usuarios en un proyecto.
 *
 * @author javier.j.menendez
 */
public final class UsermgmtUtils {

    /**
     * Constructor privado para que esta clase de utilidades no pueda ser
     * instanciada.
```

```
     */
    private UsermgmtUtils() {

    }

    /**
     * Log para mostrar informacion al usuario.
     */
    private static final Logger LOG = Logger.getLogger("UsermgmtUtils.class");

    /**
     * "Numero magico".
     */
    private static final int TRES = 3;

    /**
     * "Numero magico".
     */
    private static final int CUATRO = 4;

    /**
     * "Numero magico".
     */
    private static final int CINCO = 5;

    /**
     * Cadena con el caracter "/", usado en varias ocasiones en la clase.
     */
    private static final String BARRA = "/";

    /**
     * Cadena con el caracter "\"", usado en varias ocasiones en la clase.
     */
    private static final String COMILLAS = "\"";

    /**
     * Cadena con el caracter ",", usado en varias ocasiones en la clase.
     */
    private static final String COMA = ",";
```

```
/**
 * Cadena "usermgmt/usermgmt.txt", usada en varias ocasiones en la clase.
 */
private static final String USERMGMT_CONFFILE = "usermgmt/usermgmt.txt";

/**
 * Cadena "src/main/webapp/WEB-INF/spring/webmvc-config.xml", usada en
 * varias ocasiones en la clase.
 */
private static final String WEBMVC_CONFIG =
    "src/main/webapp/WEB-INF/spring/webmvc-config.xml";

/**
 * Cadena "usermgmt/changePassword.jspx", usada en varias ocasiones en la
 * clase.
 */
private static final String CHANGE_PASSWORD =
    "usermgmt/changePassword.jspx";

/**
 * Cadena "usermgmt/userInfo.jspx", usada en varias ocasiones en la clase.
 */
private static final String USER_INFO = "usermgmt/userInfo.jspx";

/**
 * Cadena "${usermgmt.version}", usada en varias ocasiones en la clase.
 */
private static final String USERMGMT_VERSION = "${usermgmt.version}";

/**
 * Cadena "pom.xml", usada en varias ocasiones en la clase.
 */
private static final String POM = "pom.xml";

/**
 * Cadena "src/main/resources/META-INF/spring/application.properties", usada
 * en varias ocasiones en la clase.
 */
```

```
    private static final String APPLICATION_PROPERTIES =
        "src/main/resources/META-INF/spring/application.properties";

    /**
     * Cadena
     * "src/main/resources/META-INF/spring/applicationContext-security.xml",
     * usada en varias ocasiones en la clase.
     */
    private static final String APP_CONTEXT_SEC =
        "src/main/resources/META-INF/spring/applicationContext-security.xml";

    /**
     * Cadena "src/main/resources/META-INF/spring/applicationContext.xml", usada
     * en varias ocasiones en la clase.
     */
    private static final String APPLICATION_CONTEXT =
        "src/main/resources/META-INF/spring/applicationContext.xml";

    /**
     * Cadena "usermgmt.txt", usada en varias ocasiones en la clase.
     */
    private static final String CUSTOM_USERMGMT_CONFFILE = "usermgmt.txt";

    /**
     * Cadena "hostName", usada en varias ocasiones en la clase.
     */
    private static final String HOSTNAME = "hostName";

    /**
     * Cadena "portNumber", usada en varias ocasiones en la clase.
     */
    private static final String PORT_NUMBER = "portNumber";

    /**
     * Cadena "appName", usada en varias ocasiones en la clase.
     */
    private static final String APP_NAME = "appName";

    /**
```

```
     * Cadena "-web", usada en varias ocasiones en la clase.
     */
    private static final String WEB = "-web";

    /**
     * Cadena "<security:user-service id=\"userService\">", usada en varias
     * ocasiones en la clase.
     */
    private static final String USER_SERVICE =
        "<security:user-service id=\"userService\">";

    /**
     * Cadena "filter-mapping", usada en varias ocasiones en la clase.
     */
    private static final String FILTER_MAPPING = "filter-mapping";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String BASE_PACKAGE =
        "<context:component-scan base-package=\"";

    /**
     * Configura la gestion de usuarios en el proyecto, haciendo uso de un
     * servicio remoto, mediante un modulo de gestion de usuarios contenido en
     * el mismo.
     *
     * @param internalPath Ruta de la raiz del directorio de resources del JAR
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @param cl Objeto ClassLoader necesario para acceder a los elementos
     *        contenidos en "resources" en el JAR
     */
    public static void usermgmtSetupCustom(String internalPath,
                                           String projectPath,
                                           ProjectOperations projectOperations,
                                           ClassLoader cl) {
```

```
        WebModuleUtils.copyApplicationContextSecurityToWebModule(projectPath,
            projectOperations.getFocusedProjectName());
        // Leer parametros de configuracion
        String[] params = getUserConfigurationUsermgmtSetupCustom();
        String groupId = params[0], artifactId = params[1], version = params[2],
host =
            params[TRES], name = params[CUATRO];
        // Introducir las application.properties relacionadas con la gestion de
        // usuarios
        usermgmtApplicationPropertiesCustom(host, name, projectOperations);
        // Introducir en applicationContext-security.xml el nombre del host y de
        // la aplicacion, y el numero del puerto
        applicationContextSecurityUsermgmt(projectOperations);
        // Introducir en los pom.xml las propiedades y dependencias relacionadas
        // con la gestion de usuarios
        pomsUsermgmt(groupId, artifactId, version, name, projectOperations);
        // Introducir los artifactItems correspondientes en el pom.xml del
        // modulo package
        artifactItemsPackageModule(groupId, projectOperations);
        // Introducir vistas de informacion y gestion de usuarios
        addViews(projectPath, projectOperations);
        // Introducir en webmvc-config.xml el paquete de gestion de usuarios
        // utilizado como base-package
        addUsermgmtBasePackage(projectOperations);
        // Introducir la ubicacion de los mensajes de localizacion en
        // webmvc-config.xml y applicationContext.xml:
        localizationMessagesUbication(projectOperations);
        // Realizar las acciones comunes a los 2 tipos de gestion de usuarios
        commonActions(projectOperations, projectPath, internalPath, cl);
        // Introducir en la lista de valores del bean TilesConfigurer en
        // webmvc-config.xml el valor classpath, necesario para el usermgmt
        // custom
        classpathWebmvcConfig(projectOperations);
    }

    /**
     * Configura la gestion de usuarios en el proyecto, haciendo uso de las
     * librerias de seguridad que proporciona Spring para este fin (Spring
     * Security).
```

```
     *
     * @param internalPath Ruta de la raiz del directorio de resources del JAR
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @param cl Objeto ClassLoader necesario para acceder a los elementos
     *        contenidos en "resources" en el JAR
     */
    public static void usermgmtSetupSpring(String internalPath,
                                           String projectPath,
                                           ProjectOperations projectOperations,
                                           ClassLoader cl) {

        // Copiar applicationContext-security.xml y sustituir los valores
        // configurables
        ProjectUtils
            .copyTextFile(
                projectPath,
                projectOperations.getFocusedProjectName(),
                "web/src/main/resources/META-INF/spring/applicationContext-
security.xml",
                internalPath, cl);
        configureApplicationContextSecurity(projectOperations);
        // Introducir las dependencias necesarias en el pom.xml del modulo web
        addWebModuleDeps(projectOperations);
        // Realizar las acciones comunes a los 2 tipos de gestion de usuarios
        commonActions(projectOperations, projectPath, internalPath, cl);
        // Introducir las application.properties relacionadas con la gestion de
        // usuarios
        usermgmtApplicationPropertiesSpring(projectOperations);
    }

    /**
     * Obtiene los diferentes atributos del modulo de gestion de usuarios
     * configurado por el usuario.
     *
     * @return Por orden: atributos groupId, artifactId, version del modulo de
     *         gestion de ususarios, URL de la maquina en la que se encuentra
     *         implantado el mismo, y nombre de dicho modulo
```

```
     */
    private static String[] getUserConfigurationUsermgmtSetupCustom() {

        String groupId = "", artifactId = "", version = "", host = "", name =
            "";
        BufferedReader br = null;
        try {
            br =
                new BufferedReader(new FileReader(
                    FileUtilities.getConfigurationFilePath(USERMGMT_CONFFILE)));
            groupId = br.readLine();
            artifactId = br.readLine();
            version = br.readLine();
            host = br.readLine();
            name = br.readLine();
            if (groupId == null || artifactId == null || version == null
                || host == null || name == null) {
                throw new IOException();
            }
        } catch (IOException ex) {
            LOG.log(
                Level.SEVERE,
                "ERROR. Required file \"usermgmt.txt\" not found in \"usermgmt\"
configuration directory, or with bad content");
        } finally {
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
        return new String[] {groupId, artifactId, version, host, name};
    }

    /**
     * Introduce referencias a las diferentes ubicaciones de los mensajes de
     * localizacion en los ficheros
     * "src/main/resources/META-INF/spring/applicationContext.xml" y
```

```
     * "src/main/webapp/WEB-INF/spring/webmvc-config.xml" del modulo web de un
     * proyecto.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void localizationMessagesUbication(ProjectOperations
projectOperations) {


        String oldText =
            "p:basenames=\"${home.i18n.resources},WEB-INF/i18n/application\"";
        String newText =
            "p:basenames=\"WEB-INF/i18n/messages,WEB-INF/i18n/application, WEB-
INF/i18n/portal, classpath:/META-INF/web-resources/i18n/messagesPortalContext,
classpath:/META-INF/web-resources/i18n/messagesUsermgmtClient, classpath:/META-
INF/web-resources/i18n/messagesUsermgmtContext, classpath:/META-INF/web-
resources/i18n/messagesUsermgmtAcl, classpath:/META-INF/web-
resources/i18n/service_messages\"";
        FileUtilities.replaceText(oldText, newText, WebModuleUtils
            .getWebModuleFile(APPLICATION_CONTEXT, projectOperations)
            .getAbsolutePath());
        oldText =
            "p:basenames=\"WEB-INF/i18n/messages,WEB-INF/i18n/application\"";
        FileUtilities.replaceText(oldText, newText, WebModuleUtils
            .getWebModuleFile(WEBMVC_CONFIG, projectOperations)
            .getAbsolutePath());
    }


    /**
     * Introduce en los ficheros
     * "src/main/webapp/WEB-INF/spring/webmvc-config.xml" y
     * "src/main/resources/META-INF/spring/applicationContext.xml" del modulo
     * web de un proyecto el groupId del modulo de gestion de usuarios como
     * "base-package", junto con los groupId del propio proyecto y de la
     * plataforma, que ya aparecian como tales anteriormente.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void addUsermgmtBasePackage(ProjectOperations
```

```
projectOperations) {

        try {
            String pac =
                projectOperations.getFocusedTopLevelPackage()
                    .getFullyQualifiedPackageName()
                    + COMA
                    + FileUtilities.getFirstLineConfFile("platform.txt");
            String newPac =
                FileUtilities.getFirstLineConfFile(USERMGMT_CONFFILE);
            FileUtilities.replaceText(BASE_PACKAGE + pac + COMILLAS,
                BASE_PACKAGE + pac + COMA + newPac + COMILLAS, WebModuleUtils
                    .getWebModuleFile(WEBMVC_CONFIG, projectOperations)
                    .getAbsolutePath());
            FileUtilities.replaceText(BASE_PACKAGE + pac + COMILLAS,
                BASE_PACKAGE + pac + COMA + newPac + COMILLAS, WebModuleUtils
                    .getWebModuleFile(APPLICATION_CONTEXT, projectOperations)
                    .getAbsolutePath());
        } catch (IOException ex) {
            LOG.log(
                Level.SEVERE,
                "ERROR. Required files \"usermgmt/usermgmt.txt\"
or \"platform.txt\" not found in configuration directory, or have wrong
content");
        }
    }

    /**
     * Introduce en el modulo web de un proyecto las vistas de gestion de
     * usuarios y cambio de password.
     *
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void addViews(String projectPath,
                                 ProjectOperations projectOperations) {

        String changePassword =
```

```
            FileUtilities.getConfigurationFilePath(CHANGE_PASSWORD);
        String userInfo = FileUtilities.getConfigurationFilePath(USER_INFO);
        if ((new File(changePassword)).exists()
            && (new File(userInfo)).exists()) {
            String userViews =
                WebModuleUtils.getWebModuleFile(
                    "src/main/webapp/WEB-INF/views", projectOperations)
                    .getAbsolutePath()
                    + "/users";
            if (!(new File(userViews)).mkdir()) {
                FileUtilities.problemCreatingDir(userViews);
            }
            ProjectUtils.copyExternTextFile(projectPath,
                projectOperations.getFocusedProjectName(),
                "web/src/main/webapp/WEB-INF/views/users/changePassword.jspx",
                FileUtilities.getConfigurationFilePath(CHANGE_PASSWORD));
            ProjectUtils.copyExternTextFile(projectPath,
                projectOperations.getFocusedProjectName(),
                "web/src/main/webapp/WEB-INF/views/users/userInfo.jspx",
                FileUtilities.getConfigurationFilePath(USER_INFO));
        } else {
            LOG.log(
                Level.SEVERE,
                "ERROR. Required files \"changePassword.jspx\"
and \"userInfo.jspx\" not found in \"usermgmt\" configuration directory, or have
wrong content");
        }
    }

    /**
     * Crea los "artifactItem" necesarios para el uso de un modulo de gestion de
     * usuarios, en el pom.xml del modulo package.
     *
     * @param groupId Campo groupId de los artifactItem
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void artifactItemsPackageModule(String groupId,
                                                   ProjectOperations
```

```
projectOperations) {

        try {
            addArtifactItemsPackageModule(groupId, projectOperations);
        } catch (IOException ex) {
            artifactItemsFileError();
        } catch (TransformerException e) {
            artifactItemsFileError();
        } catch (ParserConfigurationException e) {
            artifactItemsFileError();
        } catch (SAXException e) {
            artifactItemsFileError();
        } catch (XPathExpressionException e) {
            artifactItemsFileError();
        }
    }

    /**
     * Crea los "artifactItem" necesarios para el uso de un modulo de gestion de
     * usuarios, en el pom.xml del modulo package, pasando cualquier excepcion
     * que se produzca al metodo artifactItemsPackageModule.
     *
     * @param groupId Campo groupId de los artifactItem
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @throws ParserConfigurationException Si el formato del fichero pom.xml no
     *         es el adecuado
     * @throws SAXException Si el formato del fichero pom.xml no es el adecuado
     * @throws IOException Si el fichero pom.xml o el de configuracion no
     *         existen o no se puede acceder a ellos
     * @throws XPathExpressionException Si el formato del fichero pom.xml no es
     *         el adecuado
     * @throws TransformerException Si el formato del fichero pom.xml no es el
     *         adecuado
     */
    private static void addArtifactItemsPackageModule(String groupId,
                                                      ProjectOperations
projectOperations)
        throws ParserConfigurationException,
```

```
            SAXException,
            IOException,
            XPathExpressionException,
            TransformerException {

        String pomPath =
            ProjectUtils.getProjectPath(projectOperations) + BARRA
                + projectOperations.getFocusedProjectName()
                + "-package/pom.xml";
        Document doc = XmlUtils.createDocumentFromFile(pomPath);
        Node artifactItems =
            XmlUtils
                .evaluateXPath(

"/project/build/plugins/plugin/executions/execution/configuration/artifactItems"
,
                    doc).item(0);
        Element[] results = createBasicArtifactItem(doc, groupId);
        Element artifactItem = results[0], aId = results[1], type = results[2],
destFileName =
            results[TRES];
        BufferedReader br =
            new BufferedReader(new FileReader(
                FileUtilities
                    .getConfigurationFilePath("usermgmt/artifactItems.txt")));
        while (br.ready()) {
            artifactItems =
                addArtifactItem(br, aId, type, destFileName, artifactItems,
                    artifactItem);
        }
        XmlUtils.guardaXml(doc, pomPath);
    }

    /**
     * Crea un artifactItem basico que servira de plantilla para cada uno que se
     * introduzca en el fichero pom.xml del modulo web de un proyecto.
     *
     * @param doc Documento XML correspondiente al fichero pom.xml del modulo
     *        web
     * @param groupId Campo groupId comun a todos los artifactItems
```

```
     * @return Nodos artifactItem, artifactId, type y destFileName,
     *         respectivamente, correspondientes al nodo del artifactItem basico
     */
    private static Element[] createBasicArtifactItem(Document doc,
                                                     String groupId) {

        Element artifactItem = doc.createElement("artifactItem");
        Element gId = doc.createElement("groupId");
        Element aId = doc.createElement("artifactId");
        Element ver = doc.createElement("version");
        Element type = doc.createElement("type");
        Element destFileName = doc.createElement("destFileName");
        gId.setTextContent(groupId);
        ver.setTextContent(USERMGMT_VERSION);
        artifactItem.appendChild(gId);
        artifactItem.appendChild(aId);
        artifactItem.appendChild(ver);
        artifactItem.appendChild(type);
        artifactItem.appendChild(destFileName);
        return new Element[] {artifactItem, aId, type, destFileName};
    }

    /**
     * Introduce un "artifactItem" como hijo de un nodo "artifactItems".
     *
     * @param br De el se leeran los atributos del artifactItem (artifactId,
     *        type y destFileName, respectivamente
     * @param aId Nodo correspondiente al campo artifactId del artifactItem
     * @param type Nodo correspondiente al campo type del artifactItem
     * @param destFileName Nodo correspondiente al campo destFileName del
     *        artifactItem
     * @param artifactItems Nodo "artifactItems", al que se introducira como
     *        hijo el nuevo artifactItem
     * @param artifactItem Nodo tipo de artifactItem, del que "cuelgan" todos
     *        sus atributos: groupId, artifactId, version, type, destFileName
     * @return Nodo "artifactItems", con sus hijos "colgando" de el, entre los
     *         que se encuentra el nuevo artifactItem introducido
     * @throws IOException Si el archivo de configuracion del que se han de leer
     *         algunos de los campos del artifactItem no tiene un contenido
```

```
     *         apropiado
     */
    private static Node addArtifactItem(BufferedReader br, Node aId, Node type,
                                        Node destFileName, Node artifactItems,
                                        Node artifactItem) throws IOException {

        String artifactId = br.readLine();
        String typ = br.readLine();
        String destFile = br.readLine();
        if (artifactId == null || typ == null || destFile == null) {
            throw new IOException();
        }
        aId.setTextContent(artifactId);
        type.setTextContent(typ);
        destFileName.setTextContent(destFile);
        artifactItems.appendChild(artifactItem.cloneNode(true));
        return artifactItems;
    }

    /**
     * Informa al usuario de que ha habido un problema con el fichero de
     * configuracion de los artifactItems.
     */
    private static void artifactItemsFileError() {

        LOG.log(
            Level.SEVERE,
            "ERROR. Required file \"artifactItems.txt\" not found
in \"usermgmt\" configuration directory, or has wrong content");
    }

    /**
     * Introduce en los ficheros pom.xml raiz y del modulo web las dependencias
     * con respecto al modulo de gestion de usuarios.
     *
     * @param groupId groupId del modulo de gestion de usuarios
     * @param artifactId artifactId del modulo de gestion de usuarios
     * @param version version del modulo de gestion de usuarios
     * @param name Nombre del fichero war (o entregable) que se encarga de la
     *        gestion de usuarios
```

```
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void pomsUsermgmt(String groupId, String artifactId,
                                     String version, String name,
                                     ProjectOperations projectOperations) {

        try {
            // pom raiz
            String pomPath =
                ProjectUtils.getProjectPath(projectOperations) + "/pom.xml";
            PomUtils.addProperty(pomPath, "usermgmt.version", version);
            PomUtils.addProperty(pomPath, "usermgmt.delivery", name);
            // pom del modulo web
            pomPath =
                (WebModuleUtils.getWebModuleFile(POM, projectOperations))
                    .getAbsolutePath();
            PomUtils.addDependency(pomPath, groupId, artifactId,
                USERMGMT_VERSION);
        } catch (IOException ex) {
            rootPomNotFound();
        } catch (XPathExpressionException e) {
            rootPomNotFound();
        } catch (ParserConfigurationException e) {
            rootPomNotFound();
        } catch (SAXException e) {
            rootPomNotFound();
        } catch (TransformerException e) {
            rootPomNotFound();
        }
    }

    /**
     * Informa al usuario cuando no se ha encontrado el fichero pom.xml raiz de
     * un proyecto.
     */
    private static void rootPomNotFound() {
```

```
        LOG.log(Level.SEVERE,
            "ERROR. Web module's or root \"pom.xml\" file not found or has wrong
content");
    }

    /**
     * Introducer en application.properties las propiedades relacionadas con la
     * gestion de usuarios.
     *
     * @param host URL de la maquina que implementa el servicio de gestion de
     *        usuarios
     * @param name Nombre del fichero war (o entregable) que se encarga de la
     *        gestion de usuarios
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void usermgmtApplicationPropertiesCustom(String host,
                                                            String name,
                                                            ProjectOperations
projectOperations) {

        // Introducir las application.properties relacionadas con la gestion de
        // usuarios
        String appProps =
            "\nREMOTE.USERMODULE.HOST="
                + host
                + "\nUSERMGMT.NAME="
                + name
                + "\n\n"
                + FileUtilities
                    .readFileAsString(FileUtilities

.getConfigurationFilePath("usermgmt/application.properties"));
        PrintWriter out = null;
        try {
            out =
                new PrintWriter(new BufferedWriter(new FileWriter(
                    WebModuleUtils.getWebModuleFile(APPLICATION_PROPERTIES,
                        projectOperations), true)));
```

```
            out.print("\n" + appProps);
            out.close();
        } catch (IOException e) {
            LOG.log(
                Level.SEVERE,
                "ERROR. Required file \"application.properties\" not found
in \"usermgmt\" configuration directory");
        } finally {
            if (out != null) {
                out.close();
            }
        }
    }

    /**
     * Configura el fichero "applicationContext-security.xml" del proyecto para
     * el uso de un modulo de gestion de usuarios, indicado por el usuario.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void applicationContextSecurityUsermgmt(ProjectOperations
projectOperations) {

        BufferedReader confFile = null;
        try {
            String appContSecPath =
                WebModuleUtils.getWebModuleFile(APP_CONTEXT_SEC,
                    projectOperations).getAbsolutePath();
            confFile =
                new BufferedReader(new FileReader(
                    FileUtilities
                        .getConfigurationFilePath(CUSTOM_USERMGMT_CONFFILE)));
            String hostName = confFile.readLine();
            String portNumber = confFile.readLine();
            if (hostName == null || portNumber == null) {
                throw new IOException();
            }
            FileUtilities.replaceText(HOSTNAME, hostName, appContSecPath);
```

```
            FileUtilities.replaceText(PORT_NUMBER, portNumber, appContSecPath);
            FileUtilities
                .replaceText(APP_NAME,
                    projectOperations.getFocusedProjectName() + WEB,
                    appContSecPath);
        } catch (IOException ex) {
            LOG.log(
                Level.SEVERE,
                "ERROR. Required user management configuration
file \"usermgmt.txt\" not found or has wrong content");
        } finally {
            try {
                if (confFile != null) {
                    confFile.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Introduce en el fichero application.properties del proyecto las
     * propiedades relacionadas con la gestion de usuarios, segun el fichero de
     * configuracion correspondiente.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void usermgmtApplicationPropertiesSpring(ProjectOperations
projectOperations) {

        String appProps =
            FileUtilities.readFileAsString(FileUtilities
                .getConfigurationFilePath("usermgmt_application.properties"));
        PrintWriter out = null;
        try {
            out =
                new PrintWriter(new BufferedWriter(new FileWriter(
```

```
                    WebModuleUtils.getWebModuleFile(APPLICATION_PROPERTIES,
                        projectOperations), true)));
            out.print("\n" + appProps);
            out.close();
        } catch (IOException e) {
            LOG.log(
                Level.SEVERE,
                "ERROR. Required file \"usermgmt_application.properties\" not
found in configuration directory");
        } finally {
            if (out != null) {
                out.close();
            }
        }
    }

    /**
     * Introduce en el fichero pom.xml del modulo web las dependencias
     * necesarias para poder usar las librerias de gestion de usuarios de Spring
     * Security.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void addWebModuleDeps(ProjectOperations projectOperations) {

        try {
            String pomPath =
                (WebModuleUtils.getWebModuleFile(POM, projectOperations))
                    .getAbsolutePath();
            String[] aIds =
                {"spring-security-core", "spring-security-config",
                    "spring-security-web", "spring-security-cas"};
            for (int i = 0; i < aIds.length; i++) {
                PomUtils.addDependency(pomPath, "org.springframework.security",
                    aIds[i], "3.1.3.RELEASE");
            }
        } catch (IOException ex) {
            webPomNotFound();
```

```
        } catch (XPathExpressionException e) {
            webPomNotFound();
        } catch (ParserConfigurationException e) {
            webPomNotFound();
        } catch (SAXException e) {
            webPomNotFound();
        } catch (TransformerException e) {
            webPomNotFound();
        }
    }

    /**
     * Informa al usuario de que no se ha encontrado el fichero pom.xml del
     * modulo web.
     */
    private static void webPomNotFound() {

        LOG.log(Level.SEVERE,
            "ERROR. Web module's \"pom.xml\" file not found or has wrong
content");
    }

    /**
     * Establece en el fichero "applicationContext-security.xml" de un proyecto
     * la configuracion necesaria para poder utilizar la gestion de usuarios
     * proporcionada por un sistema CAS mediante las librerias de Spring
     * Security.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void configureApplicationContextSecurity(ProjectOperations
projectOperations) {

        String appContSecPath =
            WebModuleUtils.getWebModuleFile(APP_CONTEXT_SEC, projectOperations)
                .getAbsolutePath();
        replaceValuesAppContSec(projectOperations, appContSecPath);
        replaceValuesAppContSecUsers(appContSecPath);
```

```
        FileUtilities.removeBlankLines(appContSecPath);
    }

    /**
     * Introduce en el fichero "applicationContext-security.xml" los diferentes
     * parametros asociados con el sistema CAS utilizado para la gestion de
     * usuarios,en funcion de lo definido en el fichero de configuracion
     * asociado.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @param appContSecPath Ruta completa del fichero
     *        "applicationContext-security.xml"
     */
    private static void replaceValuesAppContSec(ProjectOperations
projectOperations,
                                                String appContSecPath) {

        String[] values = getValuesToReplaceAppContSec();
        String hostName = values[0], portNumber = values[1], loginUrl =
            values[2], ticketValidatorUrl = values[TRES], idAuthProvider =
            values[CUATRO], logoutUrl = values[CINCO];
        String serviceUrl =
            "http://" + hostName + ":" + portNumber + BARRA
                + projectOperations.getFocusedProjectName()
                + "-web/j_spring_cas_security_check";
        FileUtilities.replaceText("ServiceUrl", serviceUrl, appContSecPath);
        FileUtilities.replaceText("value=\"LoginUrl\"", "value=\"" + loginUrl
            + COMILLAS, appContSecPath);
        FileUtilities.replaceText("ticketValidatorUrl", ticketValidatorUrl,
            appContSecPath);
        FileUtilities.replaceText("idAuthProvider", idAuthProvider,
            appContSecPath);
        FileUtilities.replaceText("logoutUrl", logoutUrl, appContSecPath);
        FileUtilities.replaceText(HOSTNAME, hostName, appContSecPath);
        FileUtilities.replaceText(PORT_NUMBER, portNumber, appContSecPath);
        FileUtilities.replaceText(APP_NAME,
            projectOperations.getFocusedProjectName() + WEB, appContSecPath);
    }
```

```
    /**
     * Obtiene los diferentes valores que ha configurado el usuario para
     * introducir en el fichero applicationContext-security.xml.
     *
     * @return Nombre de la maquina en la que se ejecuta el sistema CAS, numero
     *         de puerto en que se ejecutara la aplicacion web (por ejemplo,
     *         tipicamente, 8080), URL de la pagina de login del CAS, URL del
     *         ticket validator del CAS, id del auth provider (unico), y URL de
     *         la pagina de logout del CAS, respectivamente
     */
    private static String[] getValuesToReplaceAppContSec() {

        String hostName = null, portNumber = null, loginUrl = null,
ticketValidatorUrl =
            null, idAuthProvider = null, logoutUrl = null;
        BufferedReader confFile = null;
        try {
            confFile =
                new BufferedReader(new FileReader(
                    FileUtilities
                        .getConfigurationFilePath(CUSTOM_USERMGMT_CONFFILE)));
            hostName = confFile.readLine();
            portNumber = confFile.readLine();
            loginUrl = confFile.readLine();
            ticketValidatorUrl = confFile.readLine();
            idAuthProvider = confFile.readLine();
            logoutUrl = confFile.readLine();
            if (hostName == null || portNumber == null || loginUrl == null
                || ticketValidatorUrl == null || idAuthProvider == null
                || logoutUrl == null) {
                throw new IOException();
            }
        } catch (IOException ex) {
            LOG.log(
                Level.SEVERE,
                "ERROR. Required user management configuration
file \"usermgmt.txt\" not found or with wrong content");
        } finally {
            try {
```

```
            if (confFile != null) {
                confFile.close();
            }
        } catch (IOException ex) {
            FileUtilities.problemClosingFile();
        }
    }
    return new String[] {hostName, portNumber, loginUrl,
        ticketValidatorUrl, idAuthProvider, logoutUrl};
}

/**
 * Introduce en el fichero "applicationContext-security.xml" los usuarios,
 * passwords y roles, a partir del fichero de comnfiguracion asociado, que
 * han de coincidir con los existentes en el sistema CAS utilizado para la
 * gestion de usuarios.
 *
 * @param appContSecPath Ruta completa del fichero
 *        "applicationContext-security.xml"
 */
private static void replaceValuesAppContSecUsers(String appContSecPath) {

    BufferedReader confFile = null;
    try {
        confFile =
            new BufferedReader(new FileReader(
                FileUtilities
                    .getConfigurationFilePath("usermgmt_users.txt")));
        StringBuffer buf = new StringBuffer();
        buf.append(USER_SERVICE);
        while (confFile.ready()) {
            String name = confFile.readLine();
            String passwd = confFile.readLine();
            String roles = confFile.readLine();
            if (name == null || passwd == null || roles == null) {
                throw new IOException();
            }
            appendValuesAppContSecUsers(buf, name, passwd, roles);
        }
```

```
            String users = buf.toString();
            FileUtilities.replaceText(USER_SERVICE, users, appContSecPath);
        } catch (IOException ex) {
            LOG.log(
                Level.SEVERE,
                "ERROR. Required user management configuration
file \"usermgmt_users.txt\" not found or with wrong content");
        } finally {
            try {
                if (confFile != null) {
                    confFile.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Introduce en un StringBuffer el nombre, password y roles de un usuario,
     * para su posterior uso por replaceValuesAppContSecUsers, a la hora de
     * configurar applicationContext-security.xml.
     *
     * @param buf StringBuffer en el que se introduciran los datos del usuario
     * @param name Nombre del usuario
     * @param passwd Password del usuario
     * @param roles Roles del usuario
     * @return El StringBuffer indicado con los datos del nuevo usuario
     *         introducidos
     */
    private static StringBuffer appendValuesAppContSecUsers(StringBuffer buf,
                                                            String name,
                                                            String passwd,
                                                            String roles) {

        buf.append("\n\t\t<security:user name=\"");
        buf.append(name);
        buf.append("\" password=\"");
        buf.append(passwd);
        buf.append("\" authorities=\"");
```

```
        buf.append(roles);
        buf.append("\"/>");
        return buf;
    }

    /**
     * Realiza las acciones comunes para la gestion de usuarios, necesarias
     * tanto cuando se usa un modulo de gestion de usuarios proporcionado por el
     * usuario, como cuando se usan las librerias de Spring Security. Estas
     * acciones incluyen: introducir filtros en el web.xml, hacer que el link de
     * logout de la web apunte a la URL correcta, y establecer redireccion
     * cuando el usuario intenta acceder con un rol que no le permite el acceso.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param internalPath Ruta de la raiz del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para obtener ficheros contenidos
     *        en "resources" en el JAR
     */
    private static void commonActions(ProjectOperations projectOperations,
                                      String projectPath, String internalPath,
                                      ClassLoader cl) {

        // Introducir filtros en el web.xml
        addFilter(projectOperations);
        // Hacer que el link de logout de la web apunte a la URL correcta
        String footer =
            WebModuleUtils.getWebModuleFile(
                "src/main/webapp/WEB-INF/views/footer.jspx", projectOperations)
                .getAbsolutePath();
        FileUtilities.replaceText("/resources/j_spring_security_logout",
            "/j_spring_security_logout", footer);
        // Establecer redireccion cuando el usuario intenta acceder con un rol
        // que no le permite el acceso
        casFailure(projectOperations, projectPath, internalPath, cl);
    }
```

```
    /**
     * Delega la implementacion de una cadena de filtros en el bean
     * springSecurityFilterChain.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void addFilter(ProjectOperations projectOperations) {

        try {
            String pomPath =
                WebModuleUtils.getWebModuleFile(
                    "src/main/webapp/WEB-INF/web.xml", projectOperations)
                    .getAbsolutePath();
            Document doc = XmlUtils.createDocumentFromFile(pomPath);
            Element filter = doc.createElement("filter");
            Element filterName = doc.createElement("filter-name");
            filterName.setTextContent("springSecurityFilterChain");
            Element filterClass = doc.createElement("filter-class");
            filterClass

.setTextContent("org.springframework.web.filter.DelegatingFilterProxy");
            Element filterMapping = doc.createElement(FILTER_MAPPING);
            Element urlPattern = doc.createElement("url-pattern");
            urlPattern.setTextContent("/*");
            filter.appendChild(filterName);
            filter.appendChild(filterClass);
            filterMapping.appendChild(filterName.cloneNode(true));
            filterMapping.appendChild(urlPattern);
            Node root = doc.getDocumentElement();
            NodeList nodes = root.getChildNodes();
            for (int i = 0; i < nodes.getLength(); i++) {
                if (FILTER_MAPPING.equals(nodes.item(i).getNodeName())) {
                    root.insertBefore(filterMapping, nodes.item(i));
                    root.insertBefore(filter, filterMapping);
                    break;
                }
            }
            XmlUtils.guardaXml(doc, pomPath);
```

```
        } catch (IOException ex) {
            webXmlNotFound();
        } catch (TransformerException e) {
            webXmlNotFound();
        } catch (ParserConfigurationException e) {
            webXmlNotFound();
        } catch (SAXException e) {
            webXmlNotFound();
        }
    }

    /**
     * Informa al usuario de que el fichero web.xml no fue encontrado o no tenia
     * el contenido adecuado.
     */
    private static void webXmlNotFound() {

        LOG.log(Level.SEVERE,
            "ERROR. \"web.xml\" file not found or has wrong content");
    }

    /**
     * Crea la pagina de fallo de autenticacion con CAS, URL "/casFailure", a la
     * que el usuario de la aplicacion desarrollada en el proyecto sera dirigido
     * cuando intente acceder a la aplicacion con un rol que no le permita el
     * acceso a la misma.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param internalPath Ruta de la raiz del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para obtener ficheros contenidos
     *        en "resources" en el JAR
     */
    private static void casFailure(ProjectOperations projectOperations,
                                   String projectPath, String internalPath,
                                   ClassLoader cl) {
```

```
        // Copiar pagina de fallo de autenticacion con CAS
        ProjectUtils.copyTextFile(projectPath,
            projectOperations.getFocusedProjectName(),
            "web/src/main/webapp/WEB-INF/views/casfailure.jspx", internalPath,
            cl);
        // Copiar vista
        try {
            copyCasFailureView(projectOperations, projectPath, internalPath,
cl);
        } catch (IOException ex) {
            viewsXmlNotFound();
        } catch (TransformerException e) {
            viewsXmlNotFound();
        } catch (SAXException e) {
            viewsXmlNotFound();
        } catch (ParserConfigurationException e) {
            viewsXmlNotFound();
        }
        // Introducir controlador
        FileUtilities
            .replaceText(
                "<mvc:view-controller path=\"/dataAccessFailure\"/>",
                "<mvc:view-controller
path=\"/dataAccessFailure\"/>\n\t<mvc:view-controller path=\"/casFailure\"/>",
                projectPath + BARRA + projectOperations.getFocusedProjectName()
                    + "-web/src/main/webapp/WEB-INF/spring/webmvc-config.xml");
    }

    /**
     * Copia la pagina de fallo de autenticacion con CAS al modulo web de un
     * pryecto.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param internalPath Ruta de la raiz del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para obtener ficheros contenidos
     *        en "resources" en el JAR
     * @throws ParserConfigurationException Si el fichero views.xml no tiene un
```

```
     *         formato correcto
     * @throws SAXException Si el fichero views.xml no tiene un formato correcto
     * @throws IOException Si el fichero views.xml no existe o no puede ser
     *         modificado
     * @throws TransformerException Si el fichero views.xml no tiene un formato
     *         correcto
     */
    private static void copyCasFailureView(ProjectOperations projectOperations,
                                           String projectPath,
                                           String internalPath, ClassLoader cl)
        throws ParserConfigurationException,
            SAXException,
            IOException,
            TransformerException {

        String viewsPath =
            projectPath + BARRA + projectOperations.getFocusedProjectName()
                + "-web/src/main/webapp/WEB-INF/views/views.xml";
        String casFailure =
            FileUtilities.readTextFile(
                "web/src/main/webapp/WEB-INF/casfailure-view.xml",
                internalPath, cl);
        Document doc = XmlUtils.createDocumentFromFile(viewsPath);
        Document casFailureView = XmlUtils.createDocumentFromString(casFailure);
        doc.getDocumentElement().appendChild(
            doc.adoptNode(casFailureView.getDocumentElement().cloneNode(true)));
        XmlUtils.guardaXml(doc, viewsPath);
        // Reintroducir en views.xml las definiciones que desaparecen al
        // poner el nuevo nodo
        FileUtilities
            .replaceText(
                "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\"?>",
                "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n<!DOCTYPE tiles-
definitions PUBLIC \"-//Apache Software Foundation//DTD Tiles Configuration
2.1//EN\" \"http://tiles.apache.org/dtds/tiles-config_2_1.dtd\">",
                viewsPath);
    }

    /**
     * Informa al usuario de que el fichero views.xml no fue encontrado o no
```

```
     * tenia el contenido adecuado.
     */
    private static void viewsXmlNotFound() {

        LOG.log(Level.SEVERE,
            "ERROR. \"views.xml\" file not found or has wrong content");
    }

    /**
     * Introduce en el fichero WEBMVC_CONFIGdel modulo web una nueva propiedad
     * para el bean tilesConfigurer, de forma que el classpath incluya el
     * fichero views.xml.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para poder determinar la ubicacion del
     *        proyecto
     */
    private static void classpathWebmvcConfig(ProjectOperations
projectOperations) {

        try {
            String path =
                WebModuleUtils.getWebModuleFile(WEBMVC_CONFIG,
                    projectOperations).getAbsolutePath();
            Document doc = XmlUtils.createDocumentFromFile(path);
            Node list =
                XmlUtils.evaluateXPath(
                    "/beans/bean[@id='tilesConfigurer']/property/list", doc)
                    .item(0);
            Node value = doc.createElement("value");
            value
                .setTextContent("classpath*:/META-INF/web-
resources/views/**/views.xml");
            list.appendChild(value);
            XmlUtils.guardaXml(doc, path);
        } catch (IOException ex) {
            webmvcConfigNotFound();
        } catch (ParserConfigurationException e) {
            webmvcConfigNotFound();
        } catch (SAXException e) {
```

```
            webmvcConfigNotFound();
        } catch (XPathExpressionException e) {
            webmvcConfigNotFound();
        } catch (TransformerException e) {
            webmvcConfigNotFound();
        }
    }

    /**
     * Informa al usuario de que no se ha encontrado el fichero
     * "webmvc-config.xml" o no tiene el formato correcto.
     */
    private static void webmvcConfigNotFound() {

        LOG.log(Level.SEVERE,
            "ERROR. File \"webmvc-config.xml\" not found or has wrong content");
    }
}
```

WebModuleUtils.java

```
package rooplusplus.roo.addon.utils;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;
import javax.xml.xpath.XPathExpressionException;

import org.springframework.roo.project.Path;
import org.springframework.roo.project.PathResolver;
import org.springframework.roo.project.ProjectOperations;
import org.w3c.dom.DOMException;
```

```
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

/**
 * Utilidades para la realizacion de diversas operaciones sobre el modulo "web"
 * de una aplicacion multimodulo desarrollada con Spring Roo.
 *
 * @author javier.j.menendez
 */
public final class WebModuleUtils {

    /**
     * Constructor privado para que esta clase de utilidades no pueda ser
     * instanciada.
     */
    private WebModuleUtils() {

    }

    /**
     * Log para mostrar informacion al usuario sobre difernetes eventos.
     */
    private static final Logger LOG = Logger.getLogger("WebModuleUtils.class");

    /**
     * Cadena con el caracter "/", usado en varias ocasiones en la clase.
     */
    private static final String BARRA = "/";

    /**
     * Cadena con el caracter comillas, usado en varias ocasiones en la clase.
     */
    private static final String COMILLAS = "\"";

    /**
     * Cadena con el caracter ":", usado en varias ocasiones en la clase.
```

```
     */
    private static final String DOS_PUNTOS = ":";

    /**
     * Cadena con el caracter ">", usado en varias ocasiones en la clase.
     */
    private static final String MAYOR_QUE = ">";

/**
     * Cadena con el caracter "<", usado en varias ocasiones en la clase.
     */
    private static final String MENOR_QUE = "<";

    /**
     * Cadena con los caracteres "']", usada en varias ocasiones en la clase.
     */
    private static final String CIERRA_COMILLA_CORCHETE = "']";

    /**
     * Cadena "mysql", usada en varias ocasiones en la clase.
     */
    private static final String MYSQL = "mysql";

    /**
     * Cadena "postgresql", usada en varias ocasiones en la clase.
     */
    private static final String POSTGRESQL = "postgresql";

    /**
     * Cadena "mssql", usada en varias ocasiones en la clase.
     */
    private static final String MSSQL = "mssql";

    /**
     * Cadena "oracle", usada en varias ocasiones en la clase.
     */
    private static final String ORACLE = "oracle";

    /**
```

```
     * Cadena "ojdbc", usada en varias ocasiones en la clase.
     */
    private static final String OJDBC = "ojdbc";

    /**
     * Cadena "jtds", usada en varias ocasiones en la clase.
     */
    private static final String JTDS = "jtds";

    /**
     * Cadena "net.sourceforge.jtds", usada en varias ocasiones en la clase.
     */
    private static final String JTDS_SOURCEFORGE = "net.sourceforge.jtds";

    /**
     * Cadena "testDS", usada en varias ocasiones en la clase.
     */
    private static final String TEST_DS = "testDS";

    /**
     * Cadena "test-web", usada en varias ocasiones en la clase.
     */
    private static final String TEST_WEB = "test-web";

    /**
     * Cadena "web", usada en varias ocasiones en la clase.
     */
    private static final String WEB = "-web";

    /**
     * Cadena "-web/pom.xml", usada en varias ocasiones en la clase.
     */
    private static final String WEB_POM = "-web/pom.xml";

    /**
     * Cadena "pom.xml", usada en varias ocasiones en la clase.
     */
    private static final String POM = "pom.xml";
```

```
/**
 * Cadena "web/src/main/resources/META-INF/spring/database.properties",
 * usada en varias ocasiones en la clase.
 */
private static final String DATABASE_PROPERTIES =
    "web/src/main/resources/META-INF/spring/database.properties";

/**
 * Cadena "version", usada en varias ocasiones en la clase.
 */
private static final String VERSION = "version";

/**
 * Cadena "-web/src/main/resources/META-INF/spring/applicationContext.xml",
 * usada en varias ocasiones en la clase.
 */
private static final String APPLICATION_CONTEXT =
    "-web/src/main/resources/META-INF/spring/applicationContext.xml";

/**
 * Cadena "/project/build/plugins/plugin/artifactId", usada en varias
 * ocasiones en la clase.
 */
private static final String ARTIFACTID =
    "/project/build/plugins/plugin/artifactId";

/**
 * Cadena "tomcat-maven-plugin", usada en varias ocasiones en la clase.
 */
private static final String TOMCAT_MAVEN_PLUGIN = "tomcat-maven-plugin";

/**
 * Cadena "jetty-maven-plugin", usada en varias ocasiones en la clase.
 */
private static final String JETTY_MAVEN_PLUGIN = "jetty-maven-plugin";

/**
 * Cadena "maven-jetty-plugin", usada en varias ocasiones en la clase.
 */
```

```
    private static final String MAVEN_JETTY_PLUGIN = "maven-jetty-plugin";

    /**
     * Cadena "configuration", usada en varias ocasiones en la clase.
     */
    private static final String CONFIGURATION = "configuration";

    /**
     * Cadena "/${project.name}", usada en varias ocasiones en la clase.
     */
    private static final String PROJECT_NAME = "/${project.name}";

    /**
     * Cadena "false", usada en varias ocasiones en la clase.
     */
    private static final String FALSE = "false";

    /**
     * Cadena "commons-dbcp", usada en varias ocasiones en la clase.
     */
    private static final String COMMONS_DBCP = "commons-dbcp";

    /**
     * Cadena "runtime", usada en varias ocasiones en la clase.
     */
    private static final String RUNTIME = "runtime";

    /**
     * Cadena "-web/src/main/", usada en varias ocasiones en la clase.
     */
    private static final String WEB_SRC_MAIN = "-web/src/main/";

    /**
     * Cadena "jdbc:oracle:thin:@", usada en varias ocasiones en la clase.
     */
    private static final String ORACLE_THIN = "jdbc:oracle:thin:@";

    /**
     * Cadena "-web/src/main/jetty/jetty.xml", usada en varias ocasiones en la
```

```
     * clase.
     */
    private static final String WEB_XML = "-web/src/main/jetty/jetty.xml";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String JAVA_VERSION_1 = "<java.version>";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String JAVA_VERSION_2 = "</java.version>";

    /**
     * Copia el fichero application.properties desde el directorio de
     * "resources" del JAR hasta el modulo "web" del proyecto en desarrollo.
     *
     * @param projectPath Path del directorio raiz del proyecto en desarrollo
     * @param internalPath Path interno del directorio de resources del JAR
     * @param projectName Nombre del proyecto en desarrollo
     * @param cl ClassLoader de la clase que invoca a este metodo, necesario `
     *        para obtener los resources del JAR
     */
    public static void copyApplicationPropertiesToWebModule(String projectPath,
                                                             String internalPath,
                                                             String projectName,
                                                             ClassLoader cl) {

        copyFileToWebModule(
            "web/src/main/resources/META-INF/spring/application.properties",
            projectPath, internalPath, projectName, cl);
        String properties =
            projectPath
                + BARRA
                + projectName
                + "-web/src/main/resources/META-
INF/spring/application.properties";
        FileUtilities.replaceText(TEST_WEB, projectName + WEB, properties);
    }
```

```
/**
 * Copia el fichero database.properties desde el directorio de "resources"
 * del JAR hasta el modulo "web" del proyecto en desarrollo.
 *
 * @param projectPath Path del directorio raiz del proyecto en desarrollo
 * @param internalPath Path interno del directorio de resources del JAR
 * @param projectName Nombre del proyecto en desarrollo
 * @param databaseName Nombre de la base de datos a utilizar por el proyecto
 *        en desarrollo
 * @param cl ClassLoader de la clase que invoca a este metodo, necesario `
 *        para obtener los resources del JAR
 */
public static void copyDatabasePropertiesToWebModule(String projectPath,
                                                      String internalPath,
                                                      String projectName,
                                                      String databaseName,
                                                      ClassLoader cl) {

    String xmlPath = projectPath + BARRA + projectName + WEB_POM;
    String pom = FileUtilities.readFileAsString(xmlPath);
    BufferedReader br = null;
    try {
        String[] dat = getDbmsXpath(pom);
        String dbms = dat[0];
        String xpath = dat[1];
        String conf = FileUtilities.getConfigurationFilePath(dbms + ".txt");
        br = new BufferedReader(new FileReader(conf));
        String version = br.readLine();
        String hibernateDialect = br.readLine();
        if (dbms == null || xpath == null || version == null
            || hibernateDialect == null) {
            throw new IOException();
        }
        hibernateDialect = "org.hibernate.dialect." + hibernateDialect;
        // Establecer el dialecto SQL correspondiente al sistema de gestion
        // de bases de datos utilizado
        newProperties(hibernateDialect, databaseName, projectName,
            projectPath, internalPath, cl);
```

```
            // Ajustar version de la base de datos (y el artifactId y el nombre,
            // en el caso de Oracle)
            databaseVersion(xmlPath, xpath, version, dbms);
        } catch (IOException ex) {
            databaseVersionError();
        } catch (XPathExpressionException e) {
            databaseVersionError();
        } catch (ParserConfigurationException e) {
            databaseVersionError();
        } catch (SAXException e) {
            databaseVersionError();
        } catch (TransformerException e) {
            databaseVersionError();
        } finally {
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Informa al usuario de que no se ha encontrado el fichero de configuracion
     * de la version del driver de la base de datos.
     */
    private static void databaseVersionError() {

        LOG.log(Level.SEVERE,
            "ERROR. Database driver version configuration file not found or
empty");
    }

    /**
     * Obtiene el nombre del sistema de gestion de bases de datos en uso por
     * parte del proyecto en desarrollo, y el XPath del nodo groupId que hace
     * referencia a dicho sistema de gestion de bases de datos.
     *
```

```
     * @param pom Contenido del fichero pom.xml del modulo web del proyecto en
     *        desarrollo
     * @return Nombre del sistema de gestion de bases de datos en uso por parte
     *         del proyecto en desarrollo, y XPath del nodo groupId que hace
     *         referencia a dicho sistema de gestion de bases de datos,
     *         respectivamente
     */
    private static String[] getDbmsXpath(String pom) {

        String dbms, xpath =
            "/project/dependencies/dependency/groupId[text()='";
        if (pom.contains(POSTGRESQL)) {
            dbms = POSTGRESQL;
            xpath = xpath + POSTGRESQL;
        } else if (pom.contains(MYSQL)) {
            dbms = MYSQL;
            xpath = xpath + MYSQL;
        } else if (pom.contains(JTDS)) {
            dbms = MSSQL;
            xpath = xpath + JTDS_SOURCEFORGE;
        } else {
            dbms = ORACLE;
            xpath = xpath + OJDBC;
        }
        xpath = xpath + CIERRA_COMILLA_CORCHETE;
        return new String[] {dbms, xpath};
    }

    /**
     * Crea el fichero database.properties para modulo web de un proyecto,
     * introduciendo en el mismo la configuracion correspondiente al sistema de
     * gestion de bases de datos utilizado.
     *
     * @param hibernateDialect Nombre de la version concreta de SQL utilizada,
     *        dependiente del sistema de gesion de bases de datos utilizado y de
     *        la version del mismo
     * @param databaseName Nombre de la base de datos a utilizar por el proyecto
     *        en desarrollo
     * @param projectName Nombre del proyecto
```

```
     * @param projectPath Ruta completa del directorio raiz del proyecto
     * @param internalPath Ruta interna de la raiz del directorio de resources
     *        del JAR
     * @param cl Objeto ClassLoader necesario para extraer el contenido de
     *        "resources" en el JAR
     */
    private static void newProperties(String hibernateDialect,
                                      String databaseName, String projectName,
                                      String projectPath, String internalPath,
                                      ClassLoader cl) {

        String newProperties =
            FileUtilities.readTextFile(DATABASE_PROPERTIES, internalPath, cl);
        newProperties =
            newProperties.replace("jndi.datasource=java:comp/env/jdbc/testDS",
                "jndi.datasource=java:comp/env/jdbc/" + projectName + "DS");
        newProperties =
            newProperties.replace(
                "database.persistence.unit=persistenceUnitTest",
                "database.persistence.unit=persistenceUnit"
                    + Character.toUpperCase(projectName.charAt(0))
                    + projectName.substring(1));
        newProperties =
            newProperties.replace("database.hibernate.default_schema=TEST",
                "database.hibernate.default_schema=" + databaseName);
        newProperties =
            newProperties.replace("HIBERNATE_DIALECT_GOES_HERE",
                hibernateDialect);
        ProjectUtils.appendTextToFile(newProperties, projectPath, projectName,
            DATABASE_PROPERTIES);

    }

    /**
     * Configura la version del sistema de gestion de bases de datos utilizado
     * en un proyecto (si este es Oracle, asigna tambien el valor apropiado del
     * groupId y artifactId del mismo), en el pom.xml indicado.
     *
     * @param xmlPath Path del fichero pom.xml en el que se configurara la
```

```
     *        version de la base de datos
     * @param xpath XPath del groupId de la dependencia de la base de datos
     *        contenida en el pom.xml
     * @param version Version del sistema de gestion de bases de datos a
     *        establecer
     * @param dbms Nombre del sistema de gestion de bases de datos a utilizar
     * @throws IOException Si xmlPath no se corresponde con un fichero pom.xml
     *         con el contenido esperado
     * @throws SAXException Si xmlPath no se corresponde con un fichero pom.xml
     *         con el contenido esperado
     * @throws ParserConfigurationException Si xmlPath no se corresponde con un
     *         fichero pom.xml con el contenido esperado
     * @throws XPathExpressionException Si xmlPath no se corresponde con un
     *         fichero pom.xml con el contenido esperado
     * @throws TransformerException Si xmlPath no se corresponde con un fichero
     *         pom.xml con el contenido esperado
     */
    private static void databaseVersion(String xmlPath, String xpath,
                                        String version, String dbms)
        throws ParserConfigurationException,
            SAXException,
            IOException,
            XPathExpressionException,
            TransformerException {

        FileUtilities.replaceText("com.oracle", OJDBC, xmlPath);
        Document doc = XmlUtils.createDocumentFromFile(xmlPath);
        Element node = (Element) XmlUtils.evaluateXPath(xpath, doc).item(0);
        Node parent = node.getParentNode();
        NodeList list = parent.getChildNodes();
        for (int i = 0; i < list.getLength(); i++) {
            Node elem = list.item(i);
            if ("artifactId".equals(elem.getNodeName()) && ORACLE.equals(dbms))
{
                elem.setTextContent(OJDBC);
            }
            if (VERSION.equals(elem.getNodeName())) {
                elem.setTextContent(version);
            }
        }
```

```
        XmlUtils.guardaXml(doc, xmlPath);
    }

    /**
     * Modifica la URL de la base de datos, en database.properties, para que
     * la maquina y el puerto sean los indicados en lugar de sus valores por
     * defecto.
     *
     * @param port Numero de puerto utilizado para el acceso a la BD
     * @param server Nombre de dominio o IP de la maquina en que se encontrara
     *        implantada la base de datos
     * @param dbms Sistema de gestion de bases de datos utilizado (oracle,
mysql,
     *        mssql, postgres)
     * @param projectPath Path del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     */
    public static void setDbPortServer(String port, String server, String dbms,
                                       String projectPath, String
projectName) {
       String dbProps = projectPath + "/" + projectName +
                        "-web/src/main/resources/META-
INF/spring/database.properties";
       FileUtilities.replaceText("//localhost", "//" + server, dbProps);
       String oldPort;
       if (dbms.equals("mysql")) {
         oldPort = "3306";
       } else if (dbms.equals("mssql")) {
         oldPort = "1433";
       } else if (dbms.equals("postgres")) {
         oldPort = "5432";
       } else {
         //Oracle
         oldPort = "1521";
       }
       FileUtilities.replaceText(":" + oldPort, ":" + port, dbProps);
    }

    /**
     * Copia el fichero "applicationContext-platform.xml" desde el directorio de
```

```
     * configuracion del usuario hasta el proyecto indicado.
     *
     * @param projectPath Path del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     */
    public static void copyApplicationContextPlatformToWebModule(String
projectPath,
                                                                  String
projectName) {

        copyConfigurationFileToWebModule(projectPath, projectName,
            "applicationContext-platform.xml",
            "src/main/resources/META-INF/spring/applicationContext-
platform.xml");
    }

    /**
     * Copia el fichero "applicationContext-security.xml" desde el directorio de
     * configuracion del usuario hasta el proyecto indicado.
     *
     * @param projectPath Path del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     */
    public static void copyApplicationContextSecurityToWebModule(String
projectPath,
                                                                  String
projectName) {

        copyConfigurationFileToWebModule(projectPath, projectName,
            "usermgmt/applicationContext-security.xml",
            "src/main/resources/META-INF/spring/applicationContext-
security.xml");
    }

    /**
     * Copia el fichero "applicationContext-portal.xml" desde el directorio de
     * configuracion del usuario hasta el proyecto indicado.
     *
     * @param projectPath Path del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     */
    public static void copyApplicationContextPortalToWebModule(String
```

```
projectPath,
                                                              String
projectName) {

        copyConfigurationFileToWebModule(projectPath, projectName,
            "applicationContext-portal.xml",
            "src/main/resources/META-INF/spring/applicationContext-portal.xml");
    }

    /**
     * Paquete el fichero header.jspx del modulo web del proyecto indicado
     * por uno adaptado al uso del portal utilizado, que el usuario ha
     * introducido en su directorio de configuracion.
     *
     * @param projectPath Path del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     */
    public static void replaceHeader(String projectPath, String projectName) {

        copyConfigurationFileToWebModule(projectPath, projectName,
            "portal-header.jspx", "src/main/webapp/WEB-INF/views/header.jspx");
    }

    /**
     * Copia un fichero desde el directorio de configuracion del usuario hasta
     * el lugar indicado del modulo web de un proyecto.
     *
     * @param projectPath Path del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     * @param pathOrigen Ruta interna del fichero origen, dentro del directorio
     *        de configuracion del usuario
     * @param pathDestino Ruta interna del destino del fichero, tomando como
     *        raiz la del modulo web
     */
    private static void copyConfigurationFileToWebModule(String projectPath,
                                                         String projectName,
                                                         String pathOrigen,
                                                         String pathDestino) {

        BufferedWriter out = null;
```

```
        try {
            String context =
                FileUtilities.readFileAsStringThrows(FileUtilities
                    .getConfigurationFilePath(pathOrigen));
            out =
                new BufferedWriter(new FileWriter(projectPath + BARRA
                    + projectName + "-web/" + pathDestino));
            out.write(context);
        } catch (IOException ex) {
            LOG.log(Level.SEVERE, "ERROR. Required file \"" + pathOrigen
                + "\" not found in configuration directory");
        } finally {
            try {
                if (out != null) {
                    out.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
    }

    /**
     * Copia un fichero desde los "resources" del JAR hasta el modulo web de un
     * proyecto.
     *
     * @param file Ruta interna del fichero a copiar dentro de los resources del
     *        JAR, que sera la misma que tenga en el destino dentro del proyecto
     * @param projectPath Path del directorio raiz del proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param projectName Nombre del proyecto
     * @param cl Objeto ClassLoader necesario para extraer el contenido de
     *        "resources" en el JAR
     */
    private static void copyFileToWebModule(String file, String projectPath,
                                            String internalPath,
                                            String projectName, ClassLoader cl)
{

        if ("".equals(projectName)) {
```

```
            return;
        }
        ProjectUtils.copyTextFile(projectPath, projectName, file, internalPath,
            cl);
    }

    /**
     * Introduce los beans messageSource, applicationPropertiesBean y
     * property-placeholder (ambos obtenidos de los "resources" del JAR) en el
     * fichero applicationContext.xml, y el ultimo de ellos tambien en
     * webmvc-config.xml dentro del modulo web de un proyecto.
     *
     * @param projectPath Path del directorio raiz del proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param projectName Nombre del proyecto
     * @param cl Objeto ClassLoader necesario para extraer el contenido de
     *        "resources" en el JAR
     */
    public static void updateAppContextWebMvcConfig(String projectPath,
                                                    String internalPath,
                                                    String projectName,
                                                    ClassLoader cl) {

        String[] filesToAppend =
            {
                "web/src/main/resources/META-INF/spring/message-source.xml",
                "web/src/main/resources/META-INF/spring/property-placeholder-
configurer.xml",
                "web/src/main/resources/META-INF/spring/application-properties-
bean.xml"};
        String xmlPath =
            projectPath + BARRA + projectName + APPLICATION_CONTEXT;
        // Eliminar el bean property-placeholder de applicationContext.xml, para
        // despues sustiruirlo por el nuevo
        XmlUtils.eliminaNodoSiExiste(xmlPath,
            "/beans/*[name()='context:property-placeholder']");
        FileUtilities
            .replaceText(
                "This will automatically locate any and all property files you
have",
```

```
                "'context:property-placeholder' will automatically locate any
and all property files you have",
                xmlPath);
        // Modificamos el applicationContext.xml introduciendo en el el
        // contenido de los otros ficheros XML
        for (int i = 0; i < filesToAppend.length; i++) {
            XmlUtils.appendFileToRoot(xmlPath, internalPath, filesToAppend[i],
                cl);
        }
        // Introducimos valores adicionales (configurados por el usuario) en la
        // lista de locations en el applicationPropertiesBean recien introducido
        addCustomValuesApplicationPropertiesBean(xmlPath);
        // Poner el nombre del proyecto en el applicationContext.xml
        FileUtilities.replaceText(TEST_WEB, projectName + WEB, xmlPath);
        // Introducir en webmvc-config.xml el property-placeholder-configurer
        xmlPath =
            projectPath + BARRA + projectName
                + "-web/src/main/webapp/WEB-INF/spring/webmvc-config.xml";
        XmlUtils.appendFileToRoot(xmlPath, internalPath, filesToAppend[1], cl);
    }

    /**
     * Introduce valores adicionales en la lista de locations en el
     * applicationPropertiesBean, dentro del fichero pom.xml del modulo web.
     *
     * @param xmlPath Ruta completa del fichero pom.xml del modulo web del
     *        proyecto en desarrollo
     */
    private static void addCustomValuesApplicationPropertiesBean(String xmlPath)
{

        String valuesToAdd =
            FileUtilities
                .readFileAsString(FileUtilities
                    .getConfigurationFilePath("locations-application-
properties.xml"));
        if (!valuesToAdd.isEmpty()) {
            FileUtilities.replaceText(
                "<value>classpath:META-INF/*.properties</value>",
                "<value>classpath:META-INF/*.properties</value>\n\t\t\t"
```

```
                    + valuesToAdd, xmlPath);
        }
    }

    /**
     * Introduce en el fichero applicationContext.xml del modulo web de un
     * proyecto el bean entityManagerFactory y el dataSource de jndi, necesarios
     * para el uso de una base de datos en el proyecto.
     *
     * @param projectPath Path del directorio raiz del proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param projectName Nombre del proyecto
     * @param cl Objeto ClassLoader necesario para extraer el contenido de
     *        "resources" en el JAR
     */
    public static void updateAppContextDatabase(String projectPath,
                                                String internalPath,
                                                String projectName,
                                                ClassLoader cl) {

        String[] filesToAppend =
            {"web/src/main/resources/META-INF/spring/jndi-datasource.xml",
                "web/src/main/resources/META-INF/spring/entity-manager-
factory.xml"};
        String xmlPath =
            projectPath + BARRA + projectName + APPLICATION_CONTEXT;
        // Primero eliminamos los nodos equivalentes a los que se introduciran
        XmlUtils
            .eliminaNodoSiExiste(
                xmlPath,

"/beans/bean[contains(@class,'org.springframework.orm.jpa.LocalContainerEntityMa
nagerFactoryBean')]");
        XmlUtils.eliminaNodoSiExiste(xmlPath,
            "/beans/bean[contains(@id,'dataSource')]");
        // Modificamos el applicationContext.xml introduciendo en el el
        // contenido de los otros ficheros XML
        for (int i = 0; i < filesToAppend.length; i++) {
            XmlUtils.appendFileToRoot(xmlPath, internalPath, filesToAppend[i],
                cl);
```

```
    }
}

/**
 * Configura los plugins de Maven para Tomcat y Jetty en el fichero pom.xml
 * del modulo web de un proyecto, introduciendo sus dependencias y numero de
 * version correspondientes, y, si el nombre del plugin para Jetty era
 * "jetty-maven-plugin", cambiandolo a "maven-jetty-plugin".
 *
 * @param driver Nombre del driver de bases de datos utilizado (ojdbc,
 *        mysql, postgresql, jtds)
 * @param pomPath Ruta completa del fichero pom.xml en el que se
 *        configuraran los plugins
 */
public static void setMavenPlugins(String driver, String pomPath) {

    try {
        doSetMavenPlugins(driver, pomPath);
    } catch (IOException ex) {
        databaseVersionError();
    } catch (XPathExpressionException e) {
        databaseVersionError();
    } catch (TransformerException e) {
        databaseVersionError();
    } catch (DOMException e) {
        databaseVersionError();
    } catch (ParserConfigurationException e) {
        databaseVersionError();
    } catch (SAXException e) {
        databaseVersionError();
    }
}

/**
 * Configura los plugins de Maven para Tomcat y Jetty en el fichero pom.xml
 * del modulo web de un proyecto, introduciendo sus dependencias y numero de
 * version correspondientes, y, si el nombre del plugin para Jetty era
 * "jetty-maven-plugin", cambiandolo a "maven-jetty-plugin". Si se producen
 * excepciones se las devuelve al metodo setMavenPlugins para que sean
```

```
     * tratadas.
     *
     * @param driver Nombre del driver de bases de datos utilizado (ojdbc,
     *        mysql, postgresql, jtds)
     * @param pomPath Ruta completa del fichero pom.xml en el que se
     *        configuraran los plugins
     * @throws IOException Si hay algun problema al configurar los plugins de
     *         tomcat y jetty
     * @throws ParserConfigurationException Si hay algun problema con el XPath
     *         del artifactId
     * @throws SAXException Si hay algun problema con el XPath del artifactId
     * @throws XPathExpressionException Si hay algun problema con el XPath del
     *         artifactId
     * @throws TransformerException Si hay algun problema con el XPath del
     *         artifactId
     */
    private static void doSetMavenPlugins(String driver, String pomPath)
        throws IOException,
            ParserConfigurationException,
            SAXException,
            XPathExpressionException,
            TransformerException {

        String[] dat = getDbData(driver);
        String groupId = dat[0], artifactId = dat[1], version = dat[2];
        Document doc = XmlUtils.createDocumentFromFile(pomPath);
        // Configurar plugins de maven para tomcat y jetty
        String xPath = ARTIFACTID;
        NodeList nodes = XmlUtils.evaluateXPath(xPath, doc);
        for (int i = 0; i < nodes.getLength(); i++) {
            if (TOMCAT_MAVEN_PLUGIN.equals(nodes.item(i).getTextContent())) {
                doc =
                    confTomcatPlugin(nodes, i, doc, pomPath, groupId,
                        artifactId, version, driver);
                nodes = XmlUtils.evaluateXPath(xPath, doc);
            }
            if (JETTY_MAVEN_PLUGIN.equals(nodes.item(i).getTextContent())
                || MAVEN_JETTY_PLUGIN.equals(nodes.item(i).getTextContent())) {
                doc =
```

```
                    confJettyPlugin(nodes, i, doc, pomPath, groupId,
                        artifactId, version, driver, xPath);
            }
        }
        if (!"".equals(driver)) {
            // Poner <scope> runtime </scope> a las dependencias (salvo que
            // no se use ningun SGBD)
            scopeRuntime(artifactId, doc, pomPath);
        }
    }

    /**
     * Obtiene los datos correspondientes al driver de bases de datos a utilizar
     * en un proyecto, segun lo que el usuario haya configurado.
     *
     * @param driver Nombre del driver de bases de datos utilizado (ojdbc,
     *        mysql, postgresql, jtds)
     * @return groupId, artifactId y version (respectivamente) del driver de
     *         bases de datos indicado, segun lo que el usuario haya indicado en
     *         el directorio de configuracion
     * @throws IOException Si el fichero de configuracion correspondiente al
     *         driver de bases de datos indicado no existe o esta vacio
     */
    private static String[] getDbData(String driver) throws IOException {

        BufferedReader br = null;
        try {
            String[] dbmsData = selectDbms(driver);
            String version, groupId = dbmsData[0], artifactId = dbmsData[1],
versionFile =
                dbmsData[2];
            br = new BufferedReader(new FileReader(versionFile));
            version = br.readLine();
            if (version == null) {
                throw new IOException();
            }
            return new String[] {groupId, artifactId, version};
        } finally {
            try {
                if (br != null) {
```

```
                br.close();
            }
        } catch (IOException ex) {
            FileUtilities.problemClosingFile();
        }
    }
}

/**
 * Obtiene los atributos de un sistema de gestion de bases de datos.
 *
 * @param driver Nombre del driver del sistema de gestion de bases de datos.
 * @return groupId, artifactId y ruta del fichero de configuracion que
 *         contiene la version del sistema de gestion de bases de datos,
 *         respectivamente
 */
private static String[] selectDbms(String driver) {

    // Config contiene el directorio raiz de configuracion del usuario
    String groupId, artifactId, versionFile, config =
        FileUtilities.getConfigurationFilePath("");
    if (POSTGRESQL.equals(driver)) {
        groupId = POSTGRESQL;
        artifactId = POSTGRESQL;
        versionFile = config + "postgresql.txt";
    } else if (MYSQL.equals(driver)) {
        groupId = MYSQL;
        artifactId = "mysql-connector-java";
        versionFile = config + "mysql.txt";
    } else if (JTDS.equals(driver)) {
        groupId = JTDS_SOURCEFORGE;
        artifactId = JTDS;
        versionFile = config + "mssql.txt";
    } else {
        groupId = OJDBC;
        artifactId = OJDBC;
        versionFile = config + "oracle.txt";
    }
    return new String[] {groupId, artifactId, versionFile};
```

```
}

/**
 * Configura, en el fichero pom.xml del modulo web de un proyecto, el plugin
 * Maven para el uso de Tomcat, introduciendole, entre otros, la dependencia
 * respecto al driver para el acceso a base de datos, si se utiliza alguna.
 *
 * @param nodes Lista de todos los nodos correspondientes a los artifactId
 *        de los diferentes plugins
 * @param i Indice del nodo correspondiente al artifactId del plugin de
 *        Tomcat, dentro de la lista anterior
 * @param doc Documento XML correspondiente al fichero pom.xml del modulo
 *        web del proyecto
 * @param pomPath Ruta completa de dicho fichero pom.xml
 * @param groupId Campo groupId de la dependencia del driver de acceso a
 *        bases de datos
 * @param artifactId Campo artifactId de la dependencia del driver de acceso
 *        a bases de datos
 * @param version Campo version de la dependencia del driver de acceso a
 *        bases de datos
 * @param driver Nombre del driver de bases de datos utilizado (ojdbc,
 *        mysql, postgresql, jtds)
 * @return Documento XML correspondiente al fichero pom.xml del modulo web
 *         del proyecto, actualizado tras realizar las diferentes
 *         modificaciones sobre el mismo
 * @throws TransformerException Si el documento pasado como parametro tiene
 *         alguna inconsistencia
 * @throws IOException Si no es posible escribir en el archivo XML indicado
 * @throws SAXException Si el nodo en la posicion i de la lista de nodos
 *         indicada no corresponde al documento pasado como parametro
 * @throws ParserConfigurationException Si el nodo en la posicion i de la
 *         lista de nodos indicada no corresponde al documento pasado como
 *         parametro, o si este tiene alguna inconsistencia
 * @throws XPathExpressionException Si el nodo en la posicion i de la lista
 *         de nodos indicada no corresponde al documento pasado como
 *         parametro
 */
private static Document confTomcatPlugin(NodeList nodes, int i,
                                         Document doc, String pomPath,
```

```
                                        String groupId, String artifactId,
                                        String version, String driver)
        throws TransformerException,
            XPathExpressionException,
            ParserConfigurationException,
            SAXException,
            IOException {

        Element conf = doc.createElement(CONFIGURATION);
        Element path = doc.createElement("path");
        path.setTextContent(PROJECT_NAME);
        Element uriEncoding = doc.createElement("uriEncoding");
        uriEncoding.setTextContent("UTF-8");
        Element additionalConfigFilesDir =
            doc.createElement("additionalConfigFilesDir");
        additionalConfigFilesDir.setTextContent("${basedir}/src/main/tomcat");
        Element systemProperties = doc.createElement("systemProperties");
        Element jmxremote =
            doc.createElement("com.sun.management.jmxremote.ssl");
        jmxremote.setTextContent(FALSE);
        systemProperties.appendChild(jmxremote);
        conf.appendChild(path);
        conf.appendChild(uriEncoding);
        conf.appendChild(additionalConfigFilesDir);
        conf.appendChild(systemProperties);
        nodes.item(i).getParentNode().appendChild(conf);
        // Guardar el resultado
        XmlUtils.guardaXml(doc, pomPath);
        Document ret = doc;
        // Introducir dependencias relacionadas con la base de datos, caso de
        // que se use
        if (!"".equals(driver)) {
            PomUtils.addDependencyToBuildPlugin(pomPath, TOMCAT_MAVEN_PLUGIN,
                groupId, artifactId, version);
            // Actualizar el documento
            ret = XmlUtils.createDocumentFromFile(pomPath);
        }
        return ret;
    }
```

```
/**
 * Configura, en el fichero pom.xml del modulo web de un proyecto, el plugin
 * Maven para el uso de Jetty, sustituyendo el nodo de configuracion
 * correspondiente, introduciendo dependencias, estableciendo la version y
 * cambiando su nombre de "jetty-maven-plugin" a "maven-jetty-plugin" (esto
 * ultimo salvo que ya hubiera sido cambiado previamente).
 *
 * @param nodeList Lista de todos los nodos correspondientes a los
 *        artifactId de los diferentes plugins
 * @param i Indice del nodo correspondiente al artifactId del plugin de
 *        Jetty, dentro de la lista anterior
 * @param doc Documento XML correspondiente al fichero pom.xml del modulo
 *        web del proyecto
 * @param pomPath Ruta completa de dicho fichero pom.xml
 * @param groupId Campo groupId de la dependencia del driver de acceso a
 *        bases de datos
 * @param artifactId Campo artifactId de la dependencia del driver de acceso
 *        a bases de datos
 * @param version Campo version de la dependencia del driver de acceso a
 *        bases de datos
 * @param driver Nombre del driver de bases de datos utilizado (ojdbc,
 *        mysql, postgresql, jtds)
 * @param xPath XPath correspondiente a la obtencion de la lista de
 *        artifactId de los diferentes plugins
 * @return Documento XML correspondiente al fichero pom.xml del modulo web
 *         del proyecto, actualizado tras realizar las diferentes
 *         modificaciones sobre el mismo
 * @throws TransformerException Si el documento pasado como parametro tiene
 *         alguna inconsistencia
 * @throws IOException Si no es posible escribir en el archivo XML indicado
 * @throws SAXException Si el nodo en la posicion i de la lista de nodos
 *         indicada no corresponde al documento pasado como parametro
 * @throws ParserConfigurationException Si el documento XML indicado tiene
 *         alguna inconsistencia
 * @throws XPathExpressionException Si el nodo en la posicion i de la lista
 *         de nodos indicada no corresponde al documento pasado como
 *         parametro
 */
```

```
private static Document confJettyPlugin(NodeList nodeList, int i,
                                        Document doc, String pomPath,
                                        String groupId, String artifactId,
                                        String version, String driver,
                                        String xPath)
    throws TransformerException,
        XPathExpressionException,
        ParserConfigurationException,
        SAXException,
        IOException {

    NodeList nodes = nodeList;
    Document ret = doc;
    NodeList list = nodes.item(i).getParentNode().getChildNodes();
    // Borramos el nodo "configuration" para sustituirlo por el nuevo, y al
    // VERSION le damos su valor correspondiente
    for (int j = 0; j < list.getLength(); j++) {
        Node node = list.item(j);
        if (node.getNodeName().equals(VERSION)) {
            node.setTextContent("6.1.18");
        }
        if (node.getNodeName().equals(CONFIGURATION)) {
            node.getParentNode().removeChild(node);
            break;
        }
    }
    jettyConfig(ret, pomPath, nodes.item(i));
    // Introducir dependencias relacionadas con la base de datos, caso de
    // que se use
    if (!"".equals(driver)) {
        PomUtils.addDependencyToBuildPlugin(pomPath, nodes.item(i)
            .getTextContent(), groupId, artifactId, version);
        PomUtils.addDependencyToBuildPlugin(pomPath, nodes.item(i)
            .getTextContent(), COMMONS_DBCP, COMMONS_DBCP, "1.4");
        // Actualizar el documento
        ret = XmlUtils.createDocumentFromFile(pomPath);
        nodes = XmlUtils.evaluateXPath(xPath, ret);
    }
    // Cambiamos el nombre del nodo de JETTY_MAVEN_PLUGIN a
```

```
        // "maven-jetty-plugin", si no se habia cambiado previamente
        if (nodes.item(i).getTextContent().equals(JETTY_MAVEN_PLUGIN)) {
            nodes.item(i).setTextContent(MAVEN_JETTY_PLUGIN);
        }
        // Guardar el resultado
        XmlUtils.guardaXml(ret, pomPath);
        return ret;
    }

    /**
     * Introduce los diferentes elementos de configuracion de Jetty en su plugin
     * Maven, en el fichero pom.xml del modulo web de un proyecto.
     *
     * @param doc Documento XML creado a partir del fichero pom.xml del modulo
     *        web del proyecto
     * @param pomPath Ruta completa de dicho fichero pom.xml
     * @param node Nodo del documento XML correspondiente al plugin Maven de
     *        Jetty, que sera el padre, por tanto, del nodo de configuracion de
     *        Jetty
     * @throws TransformerException Si no se puede escribir en el fichero XML
     *         indicado, si el documento XML es inconsistente, o si el nodo
     *         indicado pertenece a otro documento
     */
    private static void jettyConfig(Document doc, String pomPath, Node node)
        throws TransformerException {

        Element conf = doc.createElement(CONFIGURATION);
        Element scanInterval = doc.createElement("scanIntervalSeconds");
        scanInterval.setTextContent("10");
        Element jettyConfig = doc.createElement("jettyConfig");
        jettyConfig.setTextContent("src/main/jetty/jetty.xml");
        Element contextPath = doc.createElement("contextPath");
        contextPath.setTextContent(PROJECT_NAME);
        Element webAppConfig = doc.createElement("webAppConfig");
        webAppConfig.appendChild(contextPath);
        conf.appendChild(scanInterval);
        conf.appendChild(jettyConfig);
        conf.appendChild(webAppConfig);
        node.getParentNode().appendChild(conf);
```

```
        // Guardar el resultado
        XmlUtils.guardaXml(doc, pomPath);
    }

    /**
     * Introduce "runtime" como "scope" en la depencencia de un plugin respecto
     * al driver de acceso a la base de datos, y tambien a la dependencia
     * respecto a commons-dbcp, en el pom.xml del modulo web de un proyecto.
     *
     * @param artifactId Campo artifactId de la dependencia respecto al driver
     *        de la base de datos
     * @param doc Documento XML creado a partir del pom.xml del modulo web del
     *        proyecto
     * @param pomPath Ruta completa del pom.xml del modulo web del proyecto
     * @throws TransformerException Si el documento XML no contiene alguna de
     *         las 2 dependencias requeridas, no existe o no es un fichero XML
     *         valido
     * @throws XPathExpressionException Si el documento XML no contiene alguna
     *         de las 2 dependencias requeridas, no existe o no es un fichero
     *         XML valido
     */
    private static void scopeRuntime(String artifactId, Document doc,
                                     String pomPath)
        throws XPathExpressionException,
            TransformerException {

        String xPath =

"/project/build/plugins/plugin/dependencies/dependency/artifactId[text()='"
                + artifactId + CIERRA_COMILLA_CORCHETE;
        PomUtils.addScope(RUNTIME, xPath, doc, pomPath);
        xPath =

"/project/build/plugins/plugin/dependencies/dependency/artifactId[text()='common
s-dbcp']";
        PomUtils.addScope(RUNTIME, xPath, doc, pomPath);
    }

    /**
     * Cambia el nombre del fichero persistence.xml a persistenceInfo.xml,
```

```
     * dentro del modulo web de un proyecto, y establece en el mismo el valor de
     * "persistence-unit" de acuerdo al nombre del proyecto.
     *
     * @param projectPath Ruta completa del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     */
    public static void renamePersistence(String projectPath, String projectName)
{

        File persistence =
            new File(projectPath + BARRA + projectName
                + "-web/src/main/resources/META-INF/persistence.xml");
        File persistenceInfo =
            new File(projectPath + BARRA + projectName
                + "-web/src/main/resources/META-INF/persistenceInfo.xml");
        if (!persistence.renameTo(persistenceInfo)) {
            LOG.log(Level.SEVERE, "ERROR. Cannot rename \"" + persistence
                + "\" file");
        }
        FileUtilities.replaceText(
            "persistence-unit name=\"persistenceUnit\"",
            "persistence-unit name=\"persistenceUnit"
                + Character.toUpperCase(projectName.charAt(0))
                + projectName.substring(1) + COMILLAS,
            persistenceInfo.getAbsolutePath());
    }

    /**
     * Configura el plugin Maven de Eclipse del pom.xml del modulo web de un
     * proyecto para que el campo "useProjectReferences" tenga el valor "false".
     *
     * @param projectPath Ruta completa del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     */
    public static void configureEclipsePlugin(String projectPath,
                                              String projectName) {

        try {
            String pomPath = projectPath + BARRA + projectName + WEB_POM;
            Document doc = XmlUtils.createDocumentFromFile(pomPath);
```

```
            NodeList nodes = XmlUtils.evaluateXPath(ARTIFACTID, doc);
            for (int i = 0; i < nodes.getLength(); i++) {
                if (nodes.item(i).getTextContent()
                    .equals("maven-eclipse-plugin")) {
                    Element useProjectReferences =
                        doc.createElement("useProjectReferences");
                    useProjectReferences.setTextContent(FALSE);
                    NodeList hijos =
                        nodes.item(i).getParentNode().getChildNodes();
                    for (int j = 0; j < hijos.getLength(); j++) {
                        if (hijos.item(j).getNodeName().equals(CONFIGURATION)) {
                            hijos.item(j).appendChild(useProjectReferences);
                            break;
                        }
                    }
                    break;
                }
            }
            XmlUtils.guardaXml(doc, pomPath);
        } catch (IOException ex) {
            webPomNotFound();
        } catch (ParserConfigurationException e) {
            webPomNotFound();
        } catch (SAXException e) {
            webPomNotFound();
        } catch (XPathExpressionException e) {
            webPomNotFound();
        } catch (TransformerException e) {
            webPomNotFound();
        }
    }

    /**
     * Informa al usuario de que el fichero pom.xml del modulo web del proyecto
     * no ha sido encontrado o no tiene un contenido valido.
     */
    private static void webPomNotFound() {

        LOG.log(Level.SEVERE,
```

```
            "ERROR. Web module's \"pom.xml\" file not found or has wrong
content");
    }

    /**
     * Introduce en el fichero pom.xml del modulo web de un proyecto, el plugin
     * de Maven correspondiente a las licencias utilizadas.
     *
     * @param projectPath Ruta completa del directorio raiz del proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param projectName Nombre del proyecto
     * @param cl Objeto ClassLoader necesario para extraer resources del JAR
     */
    public static void addLicenseToWebModule(String projectPath,
                                             String internalPath,
                                             String projectName, ClassLoader cl)
{

        String webPomPath = projectPath + BARRA + projectName + WEB_POM;
        addBuildPlugin("web/plugin-license.xml", webPomPath, internalPath, cl);
    }

    /**
     * Introduce el contenido del fichero "plugin-source.xml" de los resources
     * del JAR, como plugin en el pom.xml del modulo web de un proyecto.
     *
     * @param webPomPath Ruta completa del fichero pom.xml del modulo web de un
     *        proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para extraer resources del JAR
     */
    public static void addSourcePlugin(String webPomPath, String internalPath,
                                       ClassLoader cl) {

        addBuildPlugin("web/plugin-source.xml", webPomPath, internalPath, cl);
    }

    /**
     * Introduce un nuevo plugin en la seccion "build/plugins" del fichero
     * pom.xml del modulo web de un proyecto, tras obtenerlo de los resouces del
```

```
     * JAR.
     *
     * @param fileToAdd Ruta interna del fichero que contiene el plugin a
     *        introducir, dentro de los resources del JAR
     * @param pomPath Ruta completa del fichero pom.xml en el que se introducira
     *        el nuevo plugin
     * @param internalPath Path interno del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para extraer resources del JAR
     */
    private static void addBuildPlugin(String fileToAdd, String pomPath,
                                       String internalPath, ClassLoader cl) {

        String text = FileUtilities.readTextFile(fileToAdd, internalPath, cl);
        try {
            Document doc = XmlUtils.createDocumentFromString(text);
            Element raizDoc = doc.getDocumentElement();
            PomUtils.addBuildPlugin(pomPath, raizDoc);
        } catch (IOException ex) {
            xmlFileNotFound(fileToAdd);
        } catch (SAXException e) {
            xmlFileNotFound(fileToAdd);
        } catch (ParserConfigurationException e) {
            xmlFileNotFound(fileToAdd);
        }
    }

    /**
     * Informa al usuario de que un fichero XML no fue encontrado o su contenido
     * no era correcto.
     *
     * @param file Ruta del fichero XML que produjo el error
     */
    private static void xmlFileNotFound(String file) {

        LOG.log(Level.SEVERE, "ERROR. \"" + file
            + "\" file not found or has wrong content");
    }

    /**
```

```
     * Copia al modulo web de un proyecto los directorios y ficheros de
     * configuracion necesarios para el despliegue del mismo con los servidores
     * web Jetty y Tomcat.
     *
     * @param projectPath Ruta completa del directorio raiz del proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param dbms Nombre del sistema de gestion de bases de datos utilizado por
     *        la base de datos (mysql, mssql, postgres u oracle)
     * @param database Nombre de la base de datos utilizada (no confundir con
     *        sistema de gestion de bases de datos)
     * @param user Nombre de usuario con el que acceder a la base de datos
     * @param password Password correspondiente a dicho usuario, para el acceso
     *        a la base de datos
     * @param server Direccion IP o nombre (incluyendo dominio, si es necesario)
     *        de la maquina en la que se encuentra la base de datos
     * @param port Puerto en el que se esta ejecutando el sistema de gestion de
     *        bases de datos en la maquina correspondiente (si se indica la
     *        cadena vacia, se asignara el numero de puerto de la instalacion
     *        por defecto del sistema de gestion de bases de datos utilizado)
     * @param projectName Nombre del proyecto
     * @param cl Objeto ClassLoader necesario para extraer resources del JAR
     */
    public static void copyJettyTomcatConf(String projectPath,
                                           String internalPath, String dbms,
                                           String database, String user,
                                           String password, String server,
                                           String port, String projectName,
                                           ClassLoader cl) {

        String[] datos = getDbDriverUrl(dbms, port, server, database);
        String databaseDriver = datos[0], databaseUrl = datos[1];
        String basePath = projectPath + BARRA + projectName + WEB_SRC_MAIN;
        File dir = new File(basePath + "tomcat");
        if (!dir.mkdir()) {
            FileUtilities.problemCreatingDir(dir.getAbsolutePath());
        }
        copyFileToWebModule("web/src/main/tomcat/context.xml", projectPath,
            internalPath, projectName, cl);
        copyFileToWebModule("web/src/main/tomcat/server.xml", projectPath,
```

```
            internalPath, projectName, cl);
        replaceContextXmlText(projectName, user, password, databaseUrl,
            databaseDriver, projectPath, internalPath, cl);
    }

    /**
     * Obtiene el nombre completo del driver y la URL de acceso para la base de
     * datos con las caracteristicas indicadas. En el caso de Oracle, obtiene
     * tambien el SID a partir de un fichero de configuracion.
     *
     * @param dbms Nombre del sistema de gestion de bases de datos utilizado por
     *        la base de datos (mysql, mssql, postgres u oracle)
     * @param dbPort Puerto en el que se esta ejecutando el sistema de gestion
     *        de bases de datos en la maquina correspondiente (si se indica la
     *        cadena vacia, se asignara el numero de puerto de la instalacion
     *        por defecto del sistema de gestion de bases de datos utilizado)
     * @param server Direccion IP o nombre (incluyendo dominio, si es necesario)
     *        de la maquina en la que se encuentra la base de datos
     * @param database Nombre de la base de datos utilizada (no confundir con
     *        sistema de gestion de bases de datos)
     * @return Nombre completo del driver de bases de datos a utilizar, y URL de
     *         la base de datos, respectivamente
     */
    private static String[] getDbDriverUrl(String dbms, String dbPort,
                                           String server, String database) {

        String databaseDriver, databaseUrl, port = dbPort;
        if (MYSQL.equals(dbms)) {
            databaseDriver = "com.mysql.jdbc.Driver";
            if ("".equals(port)) {
                port = "3306";
            }
            databaseUrl =
                "jdbc:mysql://" + server + DOS_PUNTOS + port + BARRA + database;
        } else if (MSSQL.equals(dbms)) {
            databaseDriver = "net.sourceforge.jtds.jdbc.Driver";
            if ("".equals(port)) {
                port = "1433";
            }
```

```
            databaseUrl =
                "jdbc:jtds:sqlserver://" + server + DOS_PUNTOS + port + BARRA
                    + database;
        } else if ("postgres".equals(dbms)) {
            databaseDriver = "org.postgresql.Driver";
            if ("".equals(port)) {
                port = "5432";
            }
            databaseUrl =
                "jdbc:postgresql://" + server + DOS_PUNTOS + port + BARRA
                    + database;
        } else {
            // Si la base de datos es Oracle llamamos al metodo que se encarga
            // de ello
            return getOracleDriverUrl(dbPort, server);
        }
        return new String[] {databaseDriver, databaseUrl};
    }

    /**
     * Obtiene el nombre completo del driver y la URL de acceso para la base de
     * datos Oracle con las caracteristicas indicadas, leyendo tambien el SID a
     * partir de un fichero de configuracion.
     *
     * @param dbPort Puerto en el que se esta ejecutando Oracle en la maquina
     *        correspondiente (si se indica la cadena vacia, se asignara el
     *        numero 1521, que es el puerto usado en la instalacion por defecto
     *        de Oracle)
     * @param server Direccion IP o nombre (incluyendo dominio, si es necesario)
     *        de la maquina en la que se encuentra la base de datos
     * @return Nombre completo del driver de bases de datos a utilizar, y URL de
     *         la base de datos, respectivamente
     */
    private static String[] getOracleDriverUrl(String dbPort, String server) {

        String databaseDriver, databaseUrl, port = dbPort;
        databaseDriver = "oracle.jdbc.OracleDriver";
        if ("".equals(port)) {
            port = "1521";
```

```
        }
        String conf = FileUtilities.getConfigurationFilePath("oracle_sid.txt");
        BufferedReader br = null;
        try {
            br = new BufferedReader(new FileReader(conf));
            String sid = br.readLine();
            if (sid == null) {
                throw new IOException();
            }
            databaseUrl =
                ORACLE_THIN + server + DOS_PUNTOS + port + DOS_PUNTOS + sid;
        } catch (IOException ex) {
            databaseUrl = ORACLE_THIN + server + DOS_PUNTOS + port + ":SID";
            LOG.log(Level.SEVERE,
                "ERROR. Oracle SID not found in configuration file
(\"oracle_sid.txt\")");
        } finally {
            try {
                if (br != null) {
                    br.close();
                }
            } catch (IOException ex) {
                FileUtilities.problemClosingFile();
            }
        }
        return new String[] {databaseDriver, databaseUrl};
    }

    /**
     * Establece los valores correspondientes de los diferentes campos en los
     * ficheros context.xml (de Tomcat) y jetty.xml, dentro del modulo web de un
     * proyecto.
     *
     * @param projectName Nombre del proyecto
     * @param user Nombre de usuario para el acceso a la base de datos
     * @param password Password para el acceso a la base de datos con esa cuenta
     *        de usuario
     * @param databaseUrl URL completa para el acceso a la base de datos
     * @param databaseDriver Nombre completo del driver para el acceso a la base
     *        de datos
```

```
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para extraer resources del JAR
     */
    private static void replaceContextXmlText(String projectName, String user,
                                              String password,
                                              String databaseUrl,
                                              String databaseDriver,
                                              String projectPath,
                                              String internalPath,
                                              ClassLoader cl) {

        String context =
            projectPath + BARRA + projectName
                + "-web/src/main/tomcat/context.xml";
        FileUtilities.replaceText(TEST_DS, projectName + "DS", context);
        FileUtilities.replaceText("\"user\"", COMILLAS + user + COMILLAS,
            context);
        FileUtilities.replaceText("\"password\"", COMILLAS + password
            + COMILLAS, context);
        FileUtilities.replaceText("databaseDriver", databaseDriver, context);
        FileUtilities.replaceText("databaseUrl", databaseUrl, context);
        copyJettyConf(projectPath, projectName, internalPath, cl);
        String jetty = projectPath + BARRA + projectName + WEB_XML;
        FileUtilities.replaceText(TEST_DS, projectName + "DS", jetty);
        FileUtilities
            .replaceText(">user<", MAYOR_QUE + user + MENOR_QUE, jetty);
        FileUtilities.replaceText(">password<", MAYOR_QUE + password
            + MENOR_QUE, jetty);
        FileUtilities.replaceText(">databaseDriver<", MAYOR_QUE
            + databaseDriver + MENOR_QUE, jetty);
        FileUtilities.replaceText(">databaseUrl<", MAYOR_QUE + databaseUrl
            + MENOR_QUE, jetty);
    }

    /**
     * Crea la estructura de directorios y copia los ficheros relativos a la
     * configuracion de Jetty desde los resources del JAR hasta el modulo web de
     * un proyecto, eliminando la parte relativa a la configuracion de la base
```

```
     * de datos en "jetty.xml".
     *
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para extraer resources del JAR
     */
    public static void copyJettyConfWithoutDatabase(String projectPath,
                                                    String projectName,
                                                    String internalPath,
                                                    ClassLoader cl) {

        String basePath = projectPath + BARRA + projectName + WEB_SRC_MAIN;
        String[] dirs =
            {"jetty", "jetty/etc", "jetty/logs", "jetty/webapps",
                "jetty/webapps-plus"};
        for (int i = 0; i < dirs.length; i++) {
            File dir = new File(basePath + dirs[i]);
            if (!dir.exists() && !dir.mkdir()) {
                FileUtilities.problemCreatingDir(basePath + dirs[i]);
            }
        }
        copyFileToWebModule("web/src/main/jetty/jetty.xml", projectPath,
            internalPath, projectName, cl);
        copyFileToWebModule("web/src/main/jetty/etc/realm.properties",
            projectPath, internalPath, projectName, cl);
        copyFileToWebModule("web/src/main/jetty/etc/webdefault.xml",
            projectPath, internalPath, projectName, cl);
    }

    /**
     * Crea la estructura de directorios y copia los ficheros relativos a la
     * configuracion de Jetty desde los resources del JAR hasta el modulo web de
     * un proyecto.
     *
     * @param projectPath Ruta del directorio raiz del proyecto
     * @param projectName Nombre del proyecto
     * @param internalPath Path interno del directorio de resources del JAR
     * @param cl Objeto ClassLoader necesario para extraer resources del JAR
```

```
     */
    private static void copyJettyConf(String projectPath, String projectName,
                                      String internalPath, ClassLoader cl) {

        copyJettyConfWithoutDatabase(projectPath, projectName, internalPath,
cl);
        String databaseInfo =
            FileUtilities.readTextFile("web/src/main/jetty/jetty-database.xml",
                internalPath, cl);
        FileUtilities.replaceText("</Configure>", databaseInfo
            + "\n\n</Configure>", projectPath + BARRA + projectName + WEB_XML);
    }

    /**
     * Establece en el archivo pom.xml del modulo web de un proyecto la version
     * de Java a utilizar, en formato 1.x (por ejemplo, 1.6 en lugar de 6).
     *
     * @param version Version de Java a utilizar (por ejemplo, 1.6, pero no 6)
     * @param projectOperations Objeto projectOperations de la clase que invoca
     *        al metodo, necesario para saber que proyecto tiene el foco
     */
    public static void setJavaVersion(String version,
                                      ProjectOperations projectOperations) {

        String pomPath =
            getWebModuleFile(POM, projectOperations).getAbsolutePath();
        FileUtilities.replaceText("<java.version>6</java.version>",
            JAVA_VERSION_1 + version + JAVA_VERSION_2, pomPath);
        FileUtilities.replaceText("<java.version>7</java.version>",
            JAVA_VERSION_1 + version + JAVA_VERSION_2, pomPath);
    }

    /**
     * Determina si el modulo web del proyecto que tiene el foco ya ha sido
     * configurado para el uso de una base de datos.
     *
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para saber que proyecto tiene el foco
     * @return true si el modulo web ha sido configurado para el uso de una base
     *         de datos, false en caso contrario
```

```
     */
    public static boolean isWebJpa(ProjectOperations projectOperations) {

        File f =
            WebModuleUtils.getWebModuleFile(
                "src/main/resources/META-INF/spring/database.properties",
                projectOperations);
        if (f != null && f.exists()) {
            String dbProp = FileUtilities.readFileAsString(f.getAbsolutePath());
            return dbProp.contains("jndi.datasource");
        } else {
            return false;
        }
    }

    /**
     * Establece la version de Hibernate a utilizar, en el archivo pom.xml del
     * modulo web del proyecto que tiene el foco.
     *
     * @param version Version de Hibernate a establecer
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para saber que proyecto tiene el foco
     */
    public static void setHibernateVersion(String version,
                                           ProjectOperations projectOperations)
{

        if (!version.isEmpty()) {
            String pomPath =
                WebModuleUtils.getWebModuleFile(POM, projectOperations)
                    .getAbsolutePath();
            XmlUtils
                .changeNodeValue(
                    pomPath,

"/project/dependencies/dependency/artifactId[text()='hibernate-
core']/../version",
                    version);
            XmlUtils
                .changeNodeValue(
```

```
                    pomPath,

"/project/dependencies/dependency/artifactId[text()='hibernate-
entitymanager']/../version",
                    version);


        }
    }

    /**
     * Obtiene una referencia a un fichero existente dentro del modulo web del
     * proyecto que tiene el foco.
     *
     * @param file Ruta interna del fichero, tomando como raiz la del modulo web
     * @param projectOperations Objeto ProjectOperations de la clase que invoca
     *        al metodo, necesario para saber que proyecto tiene el foco
     * @return Referencia al fichero indicado, o null si este o el modulo web no
     *         existen
     */
    public static File getWebModuleFile(String file,
                                        ProjectOperations projectOperations) {

        PathResolver pathResolver = projectOperations.getPathResolver();
        File f = new File(ProjectUtils.getProjectPath(projectOperations));
        String[] list = f.list();
        boolean encontroWeb = false;
        for (int i = 0; i < list.length && !encontroWeb; i++) {
            if (list[i].endsWith(WEB)) {
                f =
                    new File(pathResolver.getFocusedIdentifier(Path.ROOT,
                        list[i] + BARRA + file));
                encontroWeb = true;
            }
        }
        if (!encontroWeb) {
            return null;
        }
        return f;
    }
}
```

XmlUtils.java

```
package rooplusplus.roo.addon.utils;

import java.io.File;
import java.io.IOException;
import java.io.StringReader;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;

/**
 * Utilidades para el manejo de ficheros XML.
 *
 * @author javier.j.menendez
 */
public final class XmlUtils {

    /**
     * Constructor privado para que esta clase de utilidades no pueda ser
     * instanciada.
```

```
     */
    private XmlUtils() {

    }

    /**
     * Log para mostrar informacion al usuario.
     */
    private static final Logger LOG = Logger.getLogger("XmlUtils.class");

    /**
     * Cadena con el caracter ":", usado en varias ocasiones en la clase.
     */
    private static final String DOS_PUNTOS = ":";

    /**
     * Cadena con el caracter "/", usado en varias ocasiones en la clase.
     */
    private static final String BARRA = "/";

    /**
     * Cadena con el caracter "'", usado en varias ocasiones en la clase.
     */
    private static final String COMILLA = "'";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String ERROR_1 = "ERROR. File \"";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String ERROR_2 = "\" not valid XML file";

    /**
     * Cadena usada en varias ocasiones en la clase.
     */
    private static final String XMLNS = "xmlns:";
```

```
/**
 * Cadena usada en varias ocasiones en la clase.
 */
private static final String SPRING_FRAMEWORK_SCHEMA =
    "http://www.springframework.org/schema/";

/**
 * Introduce un nuevo nodo como hijo de la raiz de un documento XML.
 *
 * @param xmlPath Ruta completa del fichero XML en que se introducira el
 *        nuevo nodo
 * @param nodeName Nombre del nuevo nodo a introducir
 * @param nodeValue Contenido del nuevo nodo a introducir
 */
public static void addChildToRoot(String xmlPath, String nodeName,
                                  String nodeValue) {

    try {
        Document doc = createDocumentFromFile(xmlPath);
        Node root = doc.getDocumentElement();
        Element newNode = doc.createElement(nodeName);
        newNode.setTextContent(nodeValue);
        root.appendChild(newNode);
        // Guardar el resultado
        guardaXml(doc, xmlPath);
    } catch (IOException ex) {
        notValidXmlFile(xmlPath);
    } catch (ParserConfigurationException e) {
        notValidXmlFile(xmlPath);
    } catch (SAXException e) {
        notValidXmlFile(xmlPath);
    } catch (TransformerException e) {
        notValidXmlFile(xmlPath);
    }
}

/**
 * Introduce un nuevo nodo como hijo de otro nodo existente en un documento
```

```
     * XML.
     *
     * @param xmlPath Ruta completa del fichero XML en que se introducira el
     *        nuevo nodo
     * @param parentXPath XPath del nodo del que "colgara" el nuevo nodo
     * @param nodeName Nombre del nuevo nodo a introducir
     * @param nodeValue Contenido del nuevo nodo a introducir
     */
    public static void addChildToNode(String xmlPath, String parentXPath,
                                      String nodeName, String nodeValue) {

        try {
            Document doc = createDocumentFromFile(xmlPath);
            Node node = evaluateXPath(parentXPath, doc).item(0);
            Element newNode = doc.createElement(nodeName);
            newNode.setTextContent(nodeValue);
            node.appendChild(newNode);
            // Guardar el resultado
            guardaXml(doc, xmlPath);
        } catch (IOException ex) {
            notValidXmlFile(xmlPath);
        } catch (ParserConfigurationException e) {
            notValidXmlFile(xmlPath);
        } catch (SAXException e) {
            notValidXmlFile(xmlPath);
        } catch (TransformerException e) {
            notValidXmlFile(xmlPath);
        } catch (XPathExpressionException e) {
            notValidXmlFile(xmlPath);
        }
    }

    /**
     * Establece un nuevo valor para el contenido de un nodo en un documento
     * XML.
     *
     * @param xmlPath Ruta completa del fichero XML
     * @param nodeXPath XPath del nodo a modificar
     * @param newValue Nuevo valor para el contenido del nodo
```

```
     */
    public static void changeNodeValue(String xmlPath, String nodeXPath,
                                       String newValue) {

        try {
            Document doc = createDocumentFromFile(xmlPath);
            Node node = evaluateXPath(nodeXPath, doc).item(0);
            node.setTextContent(newValue);
            // Guardar el resultado
            guardaXml(doc, xmlPath);
        } catch (IOException ex) {
            notValidXmlFile(xmlPath);
        } catch (ParserConfigurationException e) {
            notValidXmlFile(xmlPath);
        } catch (SAXException e) {
            notValidXmlFile(xmlPath);
        } catch (TransformerException e) {
            notValidXmlFile(xmlPath);
        } catch (XPathExpressionException e) {
            notValidXmlFile(xmlPath);
        }
    }

    /**
     * Introduce un atributo para un nodo en un documento XML.
     *
     * @param xmlPath Ruta completa del fichero XML
     * @param nodeXPath XPath del nodo al que se introducira el atributo
     * @param attribute Nombre del atributo
     * @param value Valor del atributo
     */
    public static void setNodeAttribute(String xmlPath, String nodeXPath,
                                        String attribute, String value) {

        try {
            Document doc = createDocumentFromFile(xmlPath);
            Element node = (Element) evaluateXPath(nodeXPath, doc).item(0);
            node.setAttribute(attribute, value);
            // Guardar el resultado
```

```
            guardaXml(doc, xmlPath);
        } catch (IOException ex) {
            notValidXmlFile(xmlPath);
        } catch (TransformerException e) {
            notValidXmlFile(xmlPath);
        } catch (ParserConfigurationException e) {
            notValidXmlFile(xmlPath);
        } catch (SAXException e) {
            notValidXmlFile(xmlPath);
        } catch (XPathExpressionException e) {
            notValidXmlFile(xmlPath);
        }
    }

    /**
     * Informa al usuario de que el contenido de un fichero XML no es valido.
     *
     * @param xmlPath Ruta completa del fichero XML cuyo contenido no es valido
     */
    private static void notValidXmlFile(String xmlPath) {

        LOG.log(Level.SEVERE, ERROR_1 + xmlPath + ERROR_2);
    }

    /**
     * Introduce el contenido de un fichero XML perteneciente a los "resources"
     * del JAR como hijo de la raiz de otro documento del mismo tipo.
     *
     * @param xmlPath Ruta completa del fichero XML en el que se introducira el
     *        contenido del otro documento
     * @param internalPath Ruta de la raiz del directorio de resources del JAR
     * @param xmlFileToAppend Ruta del documento XML a introducir dentro del
     *        otro, partiendo de la raiz de los "resources" del JAR
     * @param cl Objeto ClassLoader necesario para obtener el fichero contenido
     *        en "resources" en el JAR
     */
    public static void appendFileToRoot(String xmlPath, String internalPath,
                                        String xmlFileToAppend, ClassLoader cl)
{
```

```
        try {
            doAppendFile(xmlPath, internalPath, xmlFileToAppend, cl);
        } catch (IOException ex) {
            appendFileToRootException(xmlPath, xmlFileToAppend);
        } catch (ParserConfigurationException e) {
            appendFileToRootException(xmlPath, xmlFileToAppend);
        } catch (SAXException e) {
            appendFileToRootException(xmlPath, xmlFileToAppend);
        } catch (TransformerException e) {
            appendFileToRootException(xmlPath, xmlFileToAppend);
        }
    }

    /**
     * Introduce el contenido de un fichero XML perteneciente a los "resources"
     * del JAR como hijo de la raiz de otro documento del mismo tipo, pasando
     * cualquier excepcion que se produzca al metodo appendFileToRoot.
     *
     * @param xmlPath Ruta completa del fichero XML en el que se introducira el
     *        contenido del otro documento
     * @param internalPath Ruta de la raiz del directorio de resources del JAR
     * @param xmlFileToAppend Ruta del documento XML a introducir dentro del
     *        otro, partiendo de la raiz de los "resources" del JAR
     * @param cl Objeto ClassLoader necesario para obtener el fichero contenido
     *        en "resources" en el JAR
     * @throws SAXException Si el fichero XML no tiene el formato adecuado
     * @throws IOException Si no se puede escribir en el fichero XML
     * @throws ParserConfigurationException Si el fichero XML no tiene el
     *         formato adecuado
     * @throws TransformerException Si el fichero XML no tiene el formato
     *         adecuado
     */
    private static void doAppendFile(String xmlPath, String internalPath,
                                     String xmlFileToAppend, ClassLoader cl)
        throws SAXException,
            IOException,
            ParserConfigurationException,
            TransformerException {
```

```
        Document doc =
            createDocumentFromString(FileUtilities.readTextFile(
                xmlFileToAppend, internalPath, cl));
        Element root = doc.getDocumentElement();
        Document config = createDocumentFromFile(xmlPath);
        String name = root.getNodeName();
        String configRoot = config.getDocumentElement().getNodeName();
        NamedNodeMap map = root.getAttributes();
        // Introducir cabeceras, y establecer el XPath del nodo para
        // eliminarlo si existe
        removeEquivalentNodeAndAddDependencies(xmlPath, name, configRoot, map);
        // Introducimos el nodo raiz del documento a introducir como hijo de
        // la raiz del otro documento
        config = createDocumentFromFile(xmlPath);
        config.getDocumentElement().appendChild(
            config.adoptNode(root.cloneNode(true)));
        // Guardar el resultado
        guardaXml(config, xmlPath);
    }

    /**
     * Tratamiento de todas las excepciones producidas en el metodo
     * appendFileToRoot.
     *
     * @param xmlPath Ruta completa del fichero XML en el que en
     *        appendFileToRoot se introduciria el contenido del otro documento.
     * @param xmlFileToAppend Ruta del documento XML a introducir dentro del
     *        otro por appendFileToRoot, partiendo de la raiz de los "resources"
     *        del JAR
     */
    private static void appendFileToRootException(String xmlPath,
                                                  String xmlFileToAppend) {

        LOG.log(Level.SEVERE, ERROR_1 + xmlPath + "\" or \"" + xmlFileToAppend
            + ERROR_2);
    }

    /**
     * Elimina el nodo equivalente al que sera introducido en un documento como
```

```
     * hijo de su raiz, creado a partir de un fichero XML. Ademas, incluye en el
     * fichero XML en el que sera introducido el nodo las dependencias
     * necesarias para su uso y el de sus atributos, si es necesario.
     *
     * @param xmlPath Ruta completa del fichero XML en el que se introducira el
     *        contenido del otro documento
     * @param name Nombre del nodo raiz del documento que sera introducido
     *        dentro del otro
     * @param configRoot Nombre del nodo raiz del documento correspondiente al
     *        fichero XML en el que sera introducido el contenido del otro
     * @param map Lista de atributos del nodo raiz del documento a introducir
     *        como hijo del nodo raiz del principal
     */
    private static void removeEquivalentNodeAndAddDependencies(String xmlPath,
                                                               String name,
                                                               String
configRoot,
                                                               NamedNodeMap map)
{

        String nodeXPath;
        if (name.contains(DOS_PUNTOS)) {
            String[] nameSplit = name.split(DOS_PUNTOS);
            nodeXPath =
                BARRA + configRoot + "/*[local-name()='" + nameSplit[1]
                    + COMILLA;
            setNodeAttribute(xmlPath, BARRA + configRoot, XMLNS + nameSplit[0],
                SPRING_FRAMEWORK_SCHEMA + nameSplit[0]);
        } else {
            nodeXPath = BARRA + configRoot + BARRA + name;
        }
        StringBuffer buf = new StringBuffer();
        for (int i = 0; i < map.getLength(); i++) {
            buf = trataAtributo(buf, map, i, xmlPath, name, configRoot);
        }
        nodeXPath = nodeXPath + buf.toString();
        if (name.contains(DOS_PUNTOS) || map.getLength() > 0) {
            nodeXPath = nodeXPath + "]";
        }
        eliminaNodoSiExiste(xmlPath, nodeXPath);
```

```
}

/**
 * Introduce en un StringBuffer la informacion XPath correspondiente a un
 * atributo de un nodo, y si el atributo contiene el caracter ":", introduce
 * en un archivo XML la dependencia correspondiente.
 *
 * @param buf StringBuffer en el que se introducira la informacion XPath
 *        correspondiente al atributo indicado
 * @param map Lista de atributos del nodo
 * @param i Indice del nodo dentro de la lista de atributos
 * @param xmlPath Ruta completa del fichero XML en el que se ha de
 *        introducir la dependencia, en caso de que el nombre del atributo
 *        contenga el caracter ":"
 * @param name Nombre del nodo al que pertenece el atributo
 * @param configRoot Nombre del nodo raiz del documento XML en el que se ha
 *        de introducir la dependencia, en caso de que el nombre del
 *        atributo contenga el caracter ":"
 * @return El StringBuffer indicado como parametro, a su vez con la parte de
 *         XPath correspondiente al atributo tratado
 */
private static StringBuffer trataAtributo(StringBuffer buf,
                                          NamedNodeMap map, int i,
                                          String xmlPath, String name,
                                          String configRoot) {

    String attrName = map.item(i).getNodeName();
    if (attrName.contains(DOS_PUNTOS)) {
        String[] attributeSplit = attrName.split(DOS_PUNTOS);
        setNodeAttribute(xmlPath, BARRA + configRoot, XMLNS
            + attributeSplit[0], SPRING_FRAMEWORK_SCHEMA
            + attributeSplit[0]);
    }
    if (name.contains(DOS_PUNTOS) || i > 0) {
        buf.append(" and @");
    } else {
        buf.append("/*[@");
    }
    buf.append(attrName);
```

```
        buf.append("='");
        buf.append(map.item(i).getTextContent());
        buf.append(COMILLA);
        return buf;
    }

    /**
     * Elimina un nodo (en caso de que exista) de un documento XML.
     *
     * @param xmlPath Ruta completa del fichero XML
     * @param nodeXPath XPath del nodo a eliminar del documento
     */
    public static void eliminaNodoSiExiste(String xmlPath, String nodeXPath) {

        try {
            Document doc = createDocumentFromFile(xmlPath);
            NodeList node = evaluateXPath(nodeXPath, doc);
            // Si el nodo no existe salimos
            if (node.getLength() == 0 || node.item(0) == null) {
                return;
            }
            node.item(0).getParentNode().removeChild(node.item(0));
            // Guardar el resultado
            guardaXml(doc, xmlPath);
        } catch (IOException ex) {
            errorRemovingNode(xmlPath);
        } catch (ParserConfigurationException e) {
            errorRemovingNode(xmlPath);
        } catch (SAXException e) {
            errorRemovingNode(xmlPath);
        } catch (XPathExpressionException e) {
            errorRemovingNode(xmlPath);
        } catch (TransformerException e) {
            errorRemovingNode(xmlPath);
        }
    }

    /**
     * Informa al usuario de que no fue posible eliminar un nodo de un documento
```

```
     * XML.
     *
     * @param xmlPath Ruta del fichero XML del que no fue posible eliminar el
     *        nodo
     */
    private static void errorRemovingNode(String xmlPath) {

        LOG.log(Level.SEVERE, "ERROR. Could not remove node from \"" + xmlPath
            + "\" document");
    }

    /**
     * Guarda el contenido de un documento XML en un fichero.
     *
     * @param doc Documento XML a guardar
     * @param xmlPath Fichero en el que se guardara el documento
     * @throws TransformerException Si no se puede escribir en el fichero
     *         indicado, o si el documento tiene un contenido inconsistente
     */
    public static void guardaXml(Document doc, String xmlPath)
        throws TransformerException {

        Transformer xformer = TransformerFactory.newInstance().newTransformer();
        xformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
        xformer.setOutputProperty(OutputKeys.INDENT, "yes");
        xformer.transform(new DOMSource(doc), new StreamResult(
            new File(xmlPath)));
        FileUtilities.removeBlankLines(xmlPath);
    }

    /**
     * Evalua una expresion XPath sobre un documento XML.
     *
     * @param xPath Expresion XPath a evaluar
     * @param doc Documento XML sobre el que se evaluara la expresion
     * @return Lista de nodos que se ajustan a lo indicado en la expresion
     *         XPath.
     * @throws XPathExpressionException Si el documento tiene un contenido
     *         inconsistente, o si no es posible evaluar la exprexion XPath
```

```
     */
    public static NodeList evaluateXPath(String xPath, Document doc)
        throws XPathExpressionException {

        return ((NodeList) (XPathFactory.newInstance().newXPath()).evaluate(
            xPath, doc, XPathConstants.NODESET));
    }

    /**
     * Crea un documento XML a partir de un archivo.
     *
     * @param xmlPath Path del archivo XML a partir del cual se creara el
     *        documento
     * @return Documento XML contenido en el archivo indicado
     * @throws ParserConfigurationException Si el archivo indicado no existe o
     *         no contiene un documento XML valido
     * @throws SAXException Si el archivo indicado no existe o no contiene un
     *         documento XML valido
     * @throws IOException Si el archivo indicado no existe o no contiene un
     *         documento XML valido
     */
    public static Document createDocumentFromFile(String xmlPath)
        throws ParserConfigurationException,
            SAXException,
            IOException {

        return DocumentBuilderFactory.newInstance().newDocumentBuilder()
            .parse(new InputSource(xmlPath));
    }

    /**
     * Crea un documento XML a partir de una cadena.
     *
     * @param str Cadena con codigo XML a partir del que se creara el documento
     * @return Documento XML construido a partir del texto de la cadena indicada
     * @throws ParserConfigurationException Si la cadena no contiene un
     *         documento XML valido
     * @throws IOException Si la cadena no contiene un documento XML valido
     * @throws SAXException Si la cadena no contiene un documento XML valido
```

```
     */
    public static Document createDocumentFromString(String str)
        throws SAXException,
            IOException,
            ParserConfigurationException {

        return DocumentBuilderFactory.newInstance().newDocumentBuilder()
            .parse(new InputSource(new StringReader(str)));
    }
}
```

package-info.java

```
/**
 * Paquete con diferentes clases de utilidades.
 */
package rooplusplus.roo.addon.utils;
```