Eusko Jaurlaritzaren Informatika Elkartea
Sociedad Informática del Gobierno Vasco

UDA - Utilidades de Desarrollo de Aplicaciones

Gestión de seguridad

Fecha: 21/03/2013 **Referencia:**

EJIE S.A.
Mediterráneo, 14
Tel. 945 01 73 00*
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01080 Vitoria-Gasteiz
www.ejie.es

Control de documentación

Título de documento: [UDA] Anexo - Gestión de seguridad

Histórico de versiones

Código:	Versión:	Fecha:	Resumen de cambios:
	1.0.0	24/06/2012	Primera versión.
	1.0.0	26/07/2012	Corrección de errata en código asociado a las *Jsps* (*EL*).
	2.1.1	09/01/2013	Cambios asociados a la optimización de las credenciales (*credentials*), al origen alternativo de las instancias del usuario y al modelo de desconexión del sistema de seguridad.
	2.2.0	21/03/2013	Obtención de credenciales, en las *jsps*, a partir de los *beans* almacenados en el contexto de *Spring*.

Cambios producidos desde la última versión

Control de difusión

Responsable: Ander Martínez

Aprobado por:

Firma: Fecha:

Distribución:

Referencias de archivo

Autor: UDA Team

Nombre archivo:

Localización:

Contenido

1. Introducción

El presente documento pretende ser una recopilación de todos los conceptos, elementos y configuraciones que rodean al sistema de seguridad de **UDA**. Dicho sistema esta concebido para cubrir las necesidades generales de seguridad, sin que los desarrolladores deban hacer grandes esfuerzos para configurarlo, y para permitir el uso de *XLNets* como cualquier otro *Authentication provider* (proveedor de autenticación).

La línea estratégica de desarrollo del módulo de seguridad de **UDA**, se basa en el uso de *Spring Security* como principal adalid de la gestión de autenticación y de autorización en las aplicaciones. De esta forma, se consigue que los desarrolladores únicamente deban conocer la protección de *url's* y métodos de la capa de servicios de negocio. Este mismo concepto se aplica cuando se requiera del uso de *XLNets* (en caso de tratarse de entornos Ejie/EJGV) en las aplicaciones. **UDA** integra, dentro de sus funcionalidades de apoyo, un módulo de gestión de *XLNets* para *Spring Security* que permite configurarlo como un *Authentication provider* (proveedor de autenticación) más.

Como cualquier otro módulo integrado dentro de la infraestructura de **UDA**, el gestor de seguridad no es un sistema cerrado ni un conjunto de elementos de obligado cumplimiento. **UDA** proporciona una configuración básica para la gestión de la seguridad y unas clases de apoyo para permitir que las aplicaciones usen *XLNets* sin tener que hacer desarrollos propios. Todo esto está destinado a facilitar la labor de los desarrolladores y a minimizar el tiempo de desarrollo en ámbitos comunes a todas las aplicaciones. En caso de que las aplicaciones requieran de necesidades específicas o no estén cubiertas todas sus expectativas en el ámbito de seguridad, podrán hacer los cambios que requieran sobre la configuración de *Spring Security* o podrán sustituir el módulo de interacción con *XLNets* (fuera del entorno Ejie/EJGV). Esto se debe a que todo el módulo de seguridad no es más que una configuración no intrusiva de *Spring Security* que puede ser modificada o sustituida bajo los estándares de diseño de *Spring* y *Spring Security*.

Para aplicaciones que estén dentro del ámbito de desarrollo del entorno de Ejie/EJGV, es importante recordar los beneficios que aporta el comunicar al grupo de “Consultoría y Áreas de Conocimiento” los cambios o modificaciones efectuados en cualquier módulo general. Gracias a este proceder, el grupo de consultoría tendrá constancia de las necesidades y los cambios efectuados, pudiendo con ello evolucionar el código de ámbito general (favoreciendo con ello a todos los desarrollos futuros) o proporcionar apoyo a la hora de desarrollar o resolver dudas funcionales.

2. Fundamentos tecnológicos

UDA basa todo el diseño del módulo de gestión de la seguridad en dos infraestructuras principales:

- ***Spring Security***, módulo del *Spring FrameWork* destinado para la gestión de seguridad de las aplicaciones.

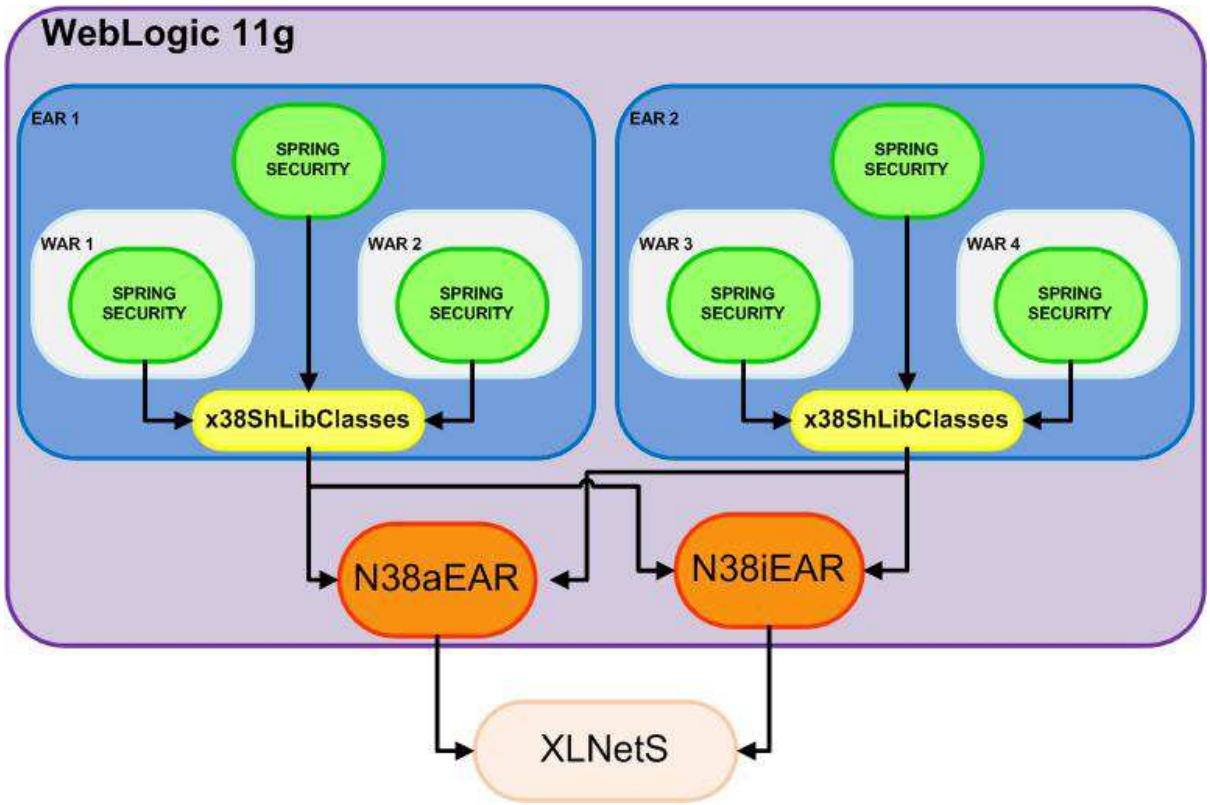
- ***XLNets***, servicios de **EJIE** y **Gobierno Vasco** destinados a la gestión de usuarios y permisos dentro del ámbito de gobierno. Dicha gestión se hace a través de la plataforma, creada para tal efecto, n38. El uso de dichos servicios es requerido en entornos asociados a Ejie/EJGV.

De esta forma, la seguridad de las aplicaciones construidas mediante las herramientas que proporciona **UDA** tendrá las siguientes características:

- Las aplicaciones delegarán la seguridad, autenticación y autorización, en *Spring Security*, con todos los beneficios que esto aporta.

- Para las aplicaciones dentro del entorno de Ejie/EJGV, el sistema de seguridad central *Java EE* continuará siendo *XLNets*.

- Será *Spring Security,* quien a través del *Wrapper* de *XLNets* que proporciona **UDA** (en la librería *x38ShLibClasses.jar*), interactuará con *XLNets* como si de cualquier otro *Authentication provider* (proveedor de autenticación) se tratase.



Aunque no es el objetivo de este documento el detallar el funcionamiento de cada uno de los elementos que conforman el modelo de seguridad, con el objetivo de orientar al lector en las funcionalidades y conceptos de cada de ellos, a continuación se introducen brevemente los más significativos.

2.1. XLNets

El sistema *XLNets* es una iniciativa horizontal de EJIE y Gobierno Vasco que busca integrar la seguridad de todos los entornos (Unix, NT, C/S…) existentes en su ámbito, a través de un único punto de acceso *web*.

La arquitectura de *XLNets* está basada en una capa cliente que integra una aplicación de *Login* y un *API* para la interacción con aplicaciones, una capa de seguridad conformada por los gestores de seguridad (gestor de sesiones, *Audit*.,…) y una capa de datos (*backend*) en el que se alberga el repositorio *LDAP*. Todas estas capas están desarrolladas con *Java* y se comunican con *XML* y un protocolo (pn38) propio del sistema.

Una de las principales funcionalidades, que nos ofrece *XLNets*, es la posibilidad de realizar controles de acceso a nuestras aplicaciones y especificar la seguridad de las mismas. Para ello es necesario realizar un análisis previo de las necesidades a nivel de seguridad de nuestra aplicación y, en función de ellas, definir la estructura del árbol de seguridad de nuestra aplicación.

Si se desea profundizar en este ámbito, se pueden consultar la documentación propia de XLNetS y más concretamente, el manual "*Definición y estudio de la seguridad de una aplicación basada en XL-Nets*" y "*XLNetS – Manual de desarrollo.doc*"

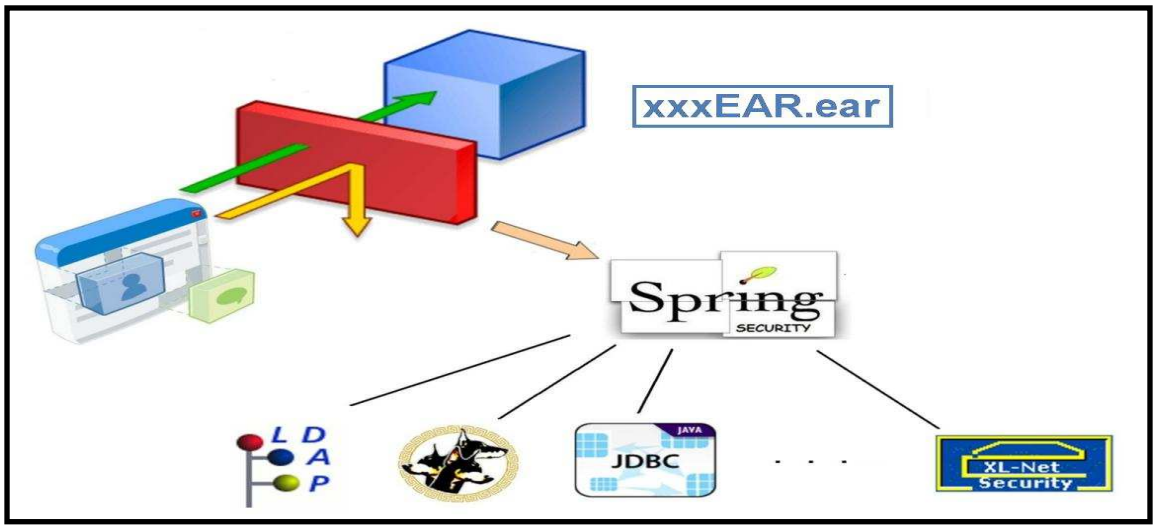
2.2. Spring Security, Authentication and Authorization Service

UDA plantea un modelo de seguridad basado en *Spring Security*. Para gestionar la seguridad en general, además de otra serie de características de seguridad, dicho *framework* trabaja con dos conceptos principales: **autenticación** y **autorización**.

Estos dos conceptos de seguridad proceden de la versión Java del *framework* estándar **PAM** (PAM, Pluggable Authentication Modules), cuyo modelo es implementado por *Spring Security*. Esta implementación introduce estos dos conceptos, en el ámbito de seguridad, de la siguiente manera:

- La **autenticación**, es la encargada de comprobar la veracidad de la identidad de cada usuario. Este sistema, independiente del medio de autenticación, se basa en la idea de presentar un servicio de autenticación separado de la tecnología utilizada para tal efecto. De esta forma, se pueden acoplar *Authentication provider* (proveedor de autenticación) nuevos o actualizados sin que se requieran cambios en la aplicación. En el caso de **UDA**, *XLNets* se ha añadido a *Spring Security* como un *Authentication provider* (proveedor de autenticación) más.

  Además, en caso de tratarse de un desarrollo no integrado en el entorno de Ejie/EJGV, la total independencia entre el sistema de seguridad y el *authentication provider* permitirá el uso de cualquier otro *provider* ajeno al propio XLNets (LDAP, Cerberus,…)



- La **autorización**, permite proteger el consumo de los distintos recursos de la aplicación a los usuarios con un nivel de acceso apropiado. Para tal efecto, *Spring Security* aporta dos posibles puntos de control que permiten a los desarrolladores imponer las restricciones de seguridad necesarias.

    - **Protección a nivel de URL**: Permite definir los permisos que debe tener un usuario para acceder a uno o a un conjunto (según patrón) de recursos Web. La configuración de este tipo de restricciones, se lleva a cabo en el fichero *security-config.xml* de cada uno de los ***Wars*** de la aplicación (más adelante, se profundiza mas en el tema).

    - **Protección a nivel de servicios de negocio**: Permite definir las credenciales de las que debe disponer un usuario para poder acceder a un servicio de negocio. Esto se configura en el fichero *security-config.xml* ubicado en el proyecto *app*EARClasses (siendo *app* el código de la aplicación) de la aplicación (más adelante, se profundiza mas sobre el tema).

2.3. Conceptos base

Antes de entrar a fondo en las configuraciones aplicadas a *Spring Security*, conviene entender algunos conceptos que rodean a *Spring Security* y hacen más comprensible su funcionamiento:

- **El objeto *Authentication***. Es la clase principal del sistema de seguridad. El objeto como tal representa la agrupación de cierta información de identificación asociada a una única entidad, por lo general una persona (el usuario que se conecta a la aplicación). El objeto está compuesto por un *Principal*, las *Credentials* asociadas a dicho *Principal* y sus *Authorities*.

    - ***Principal***. Un *Principal* representa una identidad única en el sistema. En el caso de **UDA**, y su asociación con *XLNets*, es una persona identificada por su nombre de usuario y su *UID-Session* (identificador único por cada conexión del usuario) asociado.

    - ***Credentials***. Las *credenciales* están relacionadas directamente con el *principal* y representan sus datos de identidad. Por ejemplo, *passwords*, certificados digitales, identificadores de sesión...

- ***Authorities***. Se trata de aquellos permisos que tiene el principal y que le permiten acceder a los recursos protegidos de la aplicación. La correspondencia entre *XLNets* y los roles de *Spring Security*, se hace concatenando la cadena de caracteres "***ROLE***_" a cada una de las **instancias** del usuario de *XLNets* para esa aplicación. Por ejemplo:

  Instancia: *X21A-IN-0001*

  Rol: *ROLE_X21A-IN-0001*

En caso de precisarse más información sobre los objetos de seguridad o sobre algún tipo de información complementaria asociada a *Spring Security* (otras secciones de la arquitectura de seguridad de la plataforma Java SE o Java EE), es posible acceder a la documentación oficial de *Spring Security* en las siguientes referencias:

http://static.springsource.org/spring-security/site/docs/3.1.x/reference/springsecurity.html

http://static.springframework.org/spring-security/site/index.html

2.4. Wrappers de *authentication provider*

Un a*uthentication provider* (proveedor de autenticación) es todo sistema externo, al gestor de seguridad, en el que se delega la verificación de la autenticidad del usuario y la obtención de sus autorizaciones. Este tipo de diseño, busca la independencia del gestor de seguridad, con respecto al a*uthentication provider* y permite el cambio del mismo o la migración del sistema de seguridad con cambios mínimos.

Esto es posible por que las implementaciones de gestión de seguridad, como *Spring Security*, interaccionan con el *authentication provider* a partir de un recubrimiento (wrapper) que estandariza la conectividad entre ambos.

Los gestores de seguridad, para los *authentication provider* más habituales suelen disponer de recubrimientos (*wrappers*) nativos. En el caso de *XLNets*, al ser un *authentication provider* propietario, es necesario el desarrollo de un recubrimiento (*wrapper*) específico que permita su interacción con *Spring Security*.

Para minimizar el impacto de desarrollo que implica el uso de *XLNets* con *Spring Security*, **UDA** proporciona un recubrimiento (*wrapper*) genérico que permite el uso de éste en conjunción con *Spring Security*. Mediante el uso de fachadas (*interfaces*), se implementa una solución que respeta los principios establecidos por *Spring Security* para tal efecto.

En el siguiente código *JAVA* se pueden apreciar los distintos métodos de los que dispone la fachada (*interface*) para la gestión de *Spring Security*:

```
package com.ejie.x38.security;

import java.util.HashMap;
import java.util.Vector;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
/**
 *
 * @author UDA
 *
 */
public interface PerimetralSecurityWrapper {

	public String validateSession(HttpServletRequest httpRequest, HttpServletResponse
response) throws SecurityException;

	public String getUserConnectedUserName(HttpServletRequest httpRequest);

	public HashMap<String, String> getUserDataInfo(HttpServletRequest httpRequest, boolean
isCertificate);

	public String getUserConnectedUidSession(HttpServletRequest httpRequest);

	public String getUdaValidateSessionId(HttpServletRequest httpRequest);

	public String getUserPosition(HttpServletRequest httpRequest);

	public String getURLLogin(String originalURL, boolean ajax);

	public String getPolicy(HttpServletRequest httpRequest);

	public boolean getIsCertificate(HttpServletRequest httpRequest);

	public Vector<String> getUserInstances(HttpServletRequest httpRequest);

	public void logout(HttpServletRequest httpRequest, HttpServletResponse httpResponse);

	public String getNif(HttpServletRequest httpRequest);

	public Credentials getSpecificCredentials();

	public boolean getDestroySessionSecuritySystem();

	public void setDestroySessionSecuritySystem(boolean destroySessionSecuritySystem);
}
```

Además de la propia fachada (*interface*), **UDA** proporciona dos implementaciones de la misma. Una de ellas esta asociada al uso de *XLNets* y la otra al uso de un *mock* de simulación para las ocasiones en las que no se puede trabajar directamente con *XLNets*. El posible uso de de otros sistemas de seguridad, deberían pasar por el desarrollo de sus propias implementaciones.

3. Prerrequisitos

En caso de precisar la incorporación del módulo de seguridad en cualquier aplicación *Java* EE, previamente, se precisa cumplir los siguientes prerrequisitos:

1. Si se desea utilizar como *authentication provider* a *XLNets*, el contenedor de aplicaciones deberá tener instalada la infraestructura del propio *XLNets*. Los distintos aspectos que rodean la instalación y configuración de la infraestructura de *XLNets* no serán tratados aquí ya que quedan totalmente fuera del ámbito de este documento. En caso de precisar más información sobre el tema, es posible consultar los pasos necesarios para la configuración de *XLNets* en el documento "*Instalacion_PC_local_WLS11_proveedores.v1.4.3.doc*".

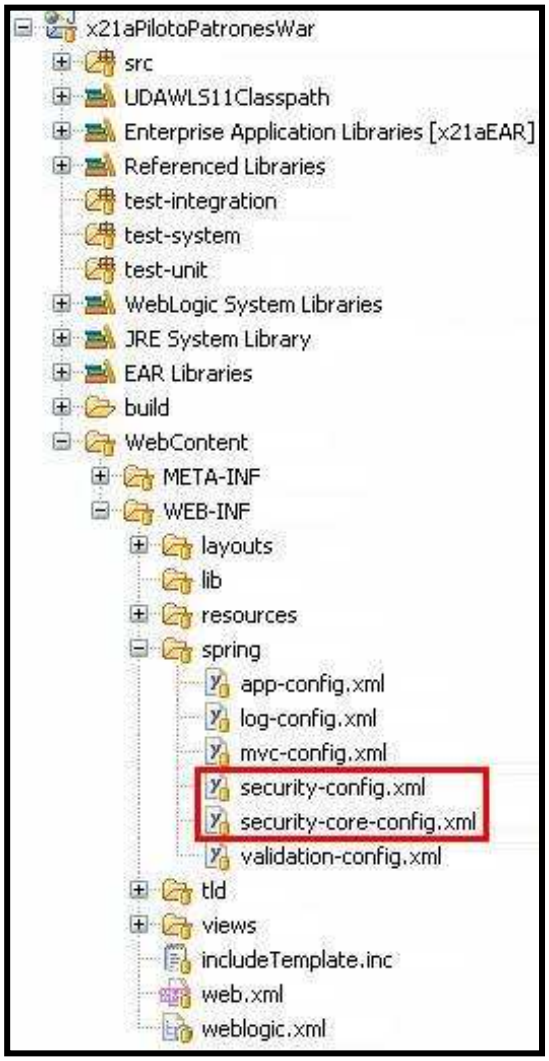
2. Para utilizar el sistema de seguridad, se ha de disponer, como mínimo, de la instalación de un contenedor de Servlets (por ejemplo *Apache Tomcat*) o de un servidor de aplicaciones (por ejemplo *JBoss*). En cualquier caso, **UDA** ha sido creado para trabajar sobre servidores de aplicaciones *WebLogic 10.3.5* (Weblogic 11g), así que, es mas recomendable el uso de dicho servidor.

3. Como en el caso del servidor, el *IDE* (entorno integrado de desarrollo) para el que se concibió **UDA** es *Eclipse*, mas concretamente, *Oracle Enterprise Pack for Eclipse 11g*. Es posible utilizar otro tipo de *IDE* (entorno integrado de desarrollo), pero, a parte de tener que ser configurados explícitamente, **UDA** ya proporciona un *Eclipse* listo para ser descargado y utilizado.

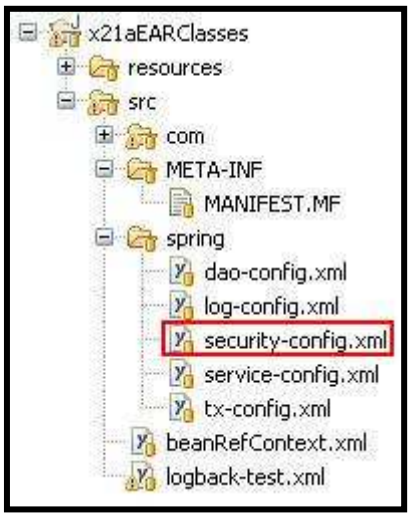
4. Infraestructura

La configuración de la seguridad en **UDA** se apoya en la inyección de dependencias, proporcionada por *Spring Framework*, para la especificación de los distintos parámetros o comportamientos de la gestión de la seguridad. Por tanto, tal y como se realiza para el resto de módulos de la aplicación, la configuración del sistema de seguridad se realiza mediante *beans* y propiedades especificados en los ficheros pensados para tal efecto.

Al igual que para el resto de módulos, los ficheros de configuración de *Spring Framework* referentes a la seguridad se han separado del resto de configuraciones para darle una división jerárquica, un poco de orden y un cierto grado de separación de conceptos. La configuración específica de seguridad se puede encontrar en los siguientes ficheros:



- ***WEB-INF/spring/security-core-config.xml*** → Contiene la configuración de los *beans* que proporcionan funcionalidad al núcleo del sistema de seguridad. Se recomienda no editar este fichero, a no ser que se sepa muy bien lo que se está haciendo, ya que podría alterar el funcionamiento general de la seguridad.
- ***WEB-INF/spring/security-config-xml*** → Contiene la configuración de seguridad para las *urls* de las *jsps* y la capa de control. En dicho fichero se aplica la seguridad de las distintas *urls* mediante expresiones regulares y los roles apropiados.
- ***spring/security-config.xml*** → Contiene la configuración de seguridad para la capa de servicios de negocio. Se especifican los servicios referenciados por expresiones regulares con los roles indicados apropiados.



Cada uno de los *wars* de la aplicación contendrá un par de ficheros *security-core-config.xml* y *security-config.xml*. Mientras que la definición de seguridad (*security-config.xml*) a nivel de servicio de lógica de negocio, se situará en el proyecto *xxxEARClasses.*

Con respecto al contenido que se va a encontrar en los distintos ficheros, a excepción del fichero *security-core-config.xml* que por su naturaleza de configuración dependiente de *Spring Security* y por la restricción de modificación que tiene, se van a describir y explicar las diferentes alternativas y estructuras que en ellos se pueden encontrar.

4.1. Configuración de seguridad a nivel de servicio

La configuración de seguridad (*security-config.xml*) presenta el siguiente aspecto:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:security="http://www.springframework.org/schema/security"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="
            http://www.springframework.org/schema/security
            http://www.springframework.org/schema/security/spring-security-3.1.xsd
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
            http://www.springframework.org/schema/util
            http://www.springframework.org/schema/util/spring-util.xsd">

<security:global-method-security>
       <security:protect-pointcut expression="execution(* com.ejie.*.service..*.find*(..))"
access="ROLE_X21A-IN-0003, ROLE_X21A-IN-0001"/>

       <security:protect-pointcut expression="execution(* com.ejie.*.service..*.add*(..))"
access="ROLE_X21A-IN-0003"/>

       <security:protect-pointcut expression="execution(* com.ejie.*.service..*.remove*(..))"
access="ROLE_X21A-IN-0003"/>

       <security:protect-pointcut expression="execution(* com.ejie.*.service..*.update*(..))"
access="ROLE_X21A-IN-0003"/>

</security:global-method-security>


</beans>
```

Como se puede apreciar el contenido del fichero es relativamente corto. Aparte de las definiciones asociadas a los *xsd* del fichero, se aprecia la configuración de seguridad sobre los distintos servicios. Mediante expresiones regulares, se declara el servicio y el método al que se le aplica la seguridad.

En general, *Spring Security* permite aplicar seguridad a nivel de clases y método. Esta característica, se aprovecha en **UDA** para dar un cierto sentido de protección a las diferentes funciones de los servicios, y así, proteger los accesos a la lógica de negocio. La restricción de acceso, viene dada por la posesión, por parte del usuario, de un tipo de rol o roles específico.

Por defecto, los métodos CRUD (findXXX(), addXXX(), removeXXX(), updateXXX()) de los servicios generados mediante las utilidades de **UDA** son sobre los que recae la seguridad. En caso de necesitar añadir nuevos métodos o clases con seguridad, o de quitar la seguridad de algún método, se puede modificar este fichero siguiendo las pautas marcadas en la documentación de *Spring Security* (http://static.springsource.org/spring-security/site/docs/3.0.x/reference/ns-config.html).

4.2. Configuración de seguridad a nivel de URL

La configuración que se puede encontrar en los ficheros *security-config*, en relación al control de las *urls*, siempre tiene una parte común y otra parte dependiente del *wrapper* que se esté utilizando en ese módulo *War*.

La parte común a todos los ficheros *security-config* de navegación, a parte de las cabeceras asociadas a los *xsd* de los ficheros, corresponde con el interceptor de seguridad que valida los permisos de los usuarios para ciertas *url*.

```
<bean id="filterSecurityInterceptor"
class="org.springframework.security.web.access.intercept.FilterSecurityInterceptor">
    <property name="authenticationManager" ref="authenticationManager" />
    <property name="accessDecisionManager" ref="affirmativeBased" />
    <property name="securityMetadataSource" >
        <security:filter-security-metadata-source use-expressions="true" request-
matcher="regex">
        <security:intercept-url pattern="/experimental/.*" access="hasRole('ROLE_X21A-IN-
0003')" />
        <security:intercept-url pattern="/**" access="isAuthenticated()" />
        </security:filter-security-metadata-source>
    </property>
</bean>
```

Los dos primeros parámetros, el *authenticationManager* y el *accessDecisionManager*, corresponden con el gestor de autenticación y el módulo que determina el acceso de un usuario o no. Esos dos parámetros, están directamente relacionados con la configuración declarada en el fichero *security-core-config.xml*, por lo que deberían tener siempre los mismos valores (así se corresponde la configuración base con las reglas de acceso especificadas para la aplicación).

La propiedad *securityMetadataSource* es el objeto encargado de recopilar todos los atributos de configuración asociados a la protección de *urls*. Dentro del *securityMetadataSource*, se define el *filter-security-metadata-source* que se encarga de recoger los filtros y que determina la manera de tratar los patrones definidos. Dichos patrones, según la opción definida en el parámetro *request-matcher*, puede ser interpretada como un patrón *path* de *ant* (*ant*), como una expresión regular (*regex*) o como una expresión regular sin considerar las mayúsculas y las minúsculas (*ciRegex*).

Los filtros aplicados a las *urls* siempre se componen de la misma manera. Por un lado, se ha de definir, según el formato del patrón definido, el literal del patrón (*pattern*), y por el otro lado, la restricción de acceso (*access*) asociada. Toda restricción se aplica a los usuarios autenticados, por lo que en caso de no estarlo, antes de comprobar la restricción de seguridad, el sistema envía al usuario a la pagina de autenticación (*login*). Las posibles restricciones de acceso que se pueden especificar son

Expresiones	**Descripciones**
hasRole([role])	Devuelve *true* si el usuario actual tiene el rol específico.
hasAnyRole([role1,role2])	Devuelve *true* si el usuario actual tiene alguno de los roles que se especifican en la lista.
nombre del usuario	Permite el acceso directo del usuario especificado.
authentication	Permite el acceso directo del usuario con que este autenticado.
permitAll	Siempre devuelve *true*.
denyAll	Siempre devuelve *false*.

isAnonymous()	Devuelve un *true* si el usuario autenticado es un usuario anónimo.
isRememberMe()	Devuelve un *true* si el usuario autenticado es un usuario *remenber-me*.
isAuthenticated()	Devuelve un *true* si el usuario autenticado no es un usuario anónimo.
isFullyAuthenticated()	Devuelve un *true* si el usuario autenticado no es un usuario anónimo ni un usuario *remenber-me*.
*hasIpAddress([*IP/NetM*])*	Devuelve un *true* si el usuario autenticado dispone de la *ip* especificada o esta dentro del rango de *ips* si se ha especificado una Netmask (por ejemplo 192.168.1.0/24 o 202.24.0.0/14).

Además de poder especificar, individualmente, cualquiera de las directivas anteriormente descritas, las restricciones de acceso también permiten el uso de algunas operaciones lógicas. Se pueden aplicar operaciones lógicas 'Y' (mediante la palabra reservada "*and*"), operaciones lógicas 'O' (mediante la palabra reservada "*or*") y negaciones (mediante la palabra reservada "*not*" o mediante el carácter reservado "!").

Algunos ejemplos de la definición de restricciones serian los siguientes:

```
<security:intercept-url pattern="/**" access="isAuthenticated()" />

<security:intercept-url pattern="/experimental/.*" access="hasRole('ROLE_X21A-IN-0003')"/>

<security:intercept-url pattern="/experimental/.*" access="hasAnyRole('ROLE_X21A-IN-
0012','ROLE_X21A-IN-0003')" />

<security:intercept-url pattern="/experimental/.*" access="hasRole('ROLE_X21A-IN-0003') and
hasIpAddress('10.170.17.0/20')"/>

<security:intercept-url pattern="/**" access="hasRole('ROLE_X21A-IN-0056') or
isAuthenticated()" />

<security:intercept-url pattern="/**" access="!hasRole('ROLE_X21A-IN-0023') and
hasIpAddress('10.170.17.0/20')" />
```

Es importante tener en cuanta que, el *accessDecisionManager,* asociado a la configuración base, pauta que para poder acceder a la *url* solicitada debe darse una verificación positiva en la primera regla que cumpla su patrón con la *url*. Esto quiere decir, que el sistema va ir probando cada una de las reglas (en orden descendente con respecto a su especificación) y en cuanto encuentra una que cumple el patrón, valida que el usuario pueda acceder o no con la restricción asociada. Esto es importante tenerlo claro por que si se ordenan mal las reglas, se puede estar dejando acceder a usuarios a recursos a los que no deberían. Un ejemplo práctico de una posible inconsistencia es el siguiente:

```
<property name="securityMetadataSource" >
  <security:filter-security-metadata-source use-expressions="true" request-matcher="regex">
    <security:intercept-url pattern="/experimental/.*" access="hasRole('ROLE_X21A-IN-0003')"/>
    <security:intercept-url pattern="/**" access="isAuthenticated()" />
  </security:filter-security-metadata-source>
</property>
```

```
<property name="securityMetadataSource" >
  <security:filter-security-metadata-source use-expressions="true" request-matcher="regex">
    <security:intercept-url pattern="/**" access="isAuthenticated()" />
    <security:intercept-url pattern="/experimental/.*" access="hasRole('ROLE_X21A-IN-0003')"/>
  </security:filter-security-metadata-source>
</property>
```

Aunque a simple vista los dos códigos no parecen muy diferentes, en el primero de los casos cuando se accede a una pagina dentro de la *suburl* "/experimental/" se valida que el usuario tenga el rol '*ROLE_X21A-IN-0003*' y en el segundo caso no. Esto se debe a que en el segundo caso, el sistema de seguridad siempre encuentra la primera restricción para validar y, una vez autenticado, cualquier usuario tendrá acceso a todas las páginas de la aplicación.

Como corolario de todo esto se puede extrae que, las reglas más restrictivas deben colocarse las primeras y las más generales al final. De esta manera, se evitará que restricciones más generales invaliden a las más específicas.

La parte dependiente del *wrapper*, varía según se este usando el *wrapper* de *XLNets* o el *wrapper* de simulación de *XLNets* denominado *mock*.

4.2.1. Wrapper de XLNets

El objetivo principal del *wrapper* de *XLNets* es permitir que *XLNets* sea un *authentication provider* más con los que trabaja *Spring Security*. Dentro de este proceso de integración se han tenido que tomar ciertas decisiones para incorporar las distintas opciones de *XLNets* a *Spring Security*, y en consecuencia a **UDA**. Todas estas decisiones se han tomado respetando *Spring Security*, se ha respetado su modelo de extensibilidad, y buscado mantener toda la potencia funcional de *XLNets*.

De todo el desarrollo del *wrapper* han surgido una serie de parámetros o configuraciones destinadas a cubrir todas las posibilidades funcionales de *XLNets* (tanto desde el punto de vista de *XLNets* como desde el punto de vista de *Spring* y de *Spring Security*). Los distintos parámetros que se pueden, o deben, utilizar son los siguientes:

```
<!-- Definicion del Wrapper de seguridad utilizado -->
<bean id="perimetralSecurityWrapper"
class="com.ejie.x38.security.PerimetralSecurityWrapperN38Impl">
       <property name="xlnetCachingPeriod" value="120" />
       <property name="specificCredentials" ref="aplicationSpecificCredentials"/>
       <property name="userChangeUrl" value="/x21aPilotoPatronesWar/" />
       <property name="alternativeStorageUserCredentials" ref="jdbcUserService"/>
       <property name="anonymousCredentials">
              <map>
                     <entry key="userProfiles" value="udaAnonymousProfile" />
                     <entry key="position" value="udaAnonymousPosition" />
              </map>
       </property>
       <property name="alternativeOriginCredentialsApp" value="y52b"></property>
       <property name="excludeFilter" ref="usuariosConCertificado"/>
</bean>

<bean id="aplicationSpecificCredentials" class="com.ejie.x21a.security.myUserCredentials"/>

<!-- Se especifica un filtro de exclusión sobre los usuarios validados. Esto permite la
limitación del acceso aunque se han usuarios validos de XLNets. -->
<bean id="usuariosConCertificado" class="com.ejie.x21a.security.UsuariosConCertificado">
       <property name="accessDeniedUrl" value="/x21aPilotoPatronesWar/accessDenied" />
</bean>
```

```
<!-- A conctinuación, se parametrizan los distintos aspectos asociada al manejo de
credenciales mediante una base de datos en Spring Security  -->
<bean id="jdbcUserService" class="com.ejie.x21a.security.SecurityCustomJdbcDaoImpl">
	<property name="dataSource" ref="dataSource"/>
	<property name="positionByUserdataQuery">
		<value> select PUESTO from CERT_USER_DATA where UPPER(USERNAME) = ? and
UPPER(DNI) = ? </value>
	</property>
	<property name="authoritiesByUserdataQuery">
		<value> select ROLES from CERT_USER_ROLES where UPPER(USERNAME) = ? and
UPPER(DNI) = ? </value>
	</property>
</bean>
```

Para explicar mejor los distintos aspectos de configuración, se van a ir extrayendo diferentes fragmentos del código y explicándolos de forma encadenada.

```
<!-- Definicion del Wrapper de seguridad utilizado -->
<bean id="perimetralSecurityWrapper"
class="com.ejie.x38.security.PerimetralSecurityWrapperN38Impl">
	<property name="xlnetCachingPeriod" value="120" />
	<property name="userChangeUrl" value="/x21aPilotoPatronesWar/" />
	<property name="alternativeStorageUserCredentials" ref="jdbcUserService"/>
	<property name="anonymousCredentials">
		<map>
			<entry key="userProfiles" value="udaAnonymousProfile" />
			<entry key="position" value="udaAnonymousPosition" />
		</map>
	</property>
	<property name="excludeFilter" ref="usuariosConCertificado"/>
</bean>
```

Esta primera parte de código representa la especificación base del *wrapper*. El punto de enlace entre la configuración general y el w*rapper* utilizado es el *bean* "*perimetralSecurityWrapper*". A dicho *bean* se le asocia la clase "*com.ejie.x38.security.PerimetralSecurityWrapperN38Impl*" que denota el uso del *wrapper* de *XLNets* (físicamente es la clase que lo implementa).

Una vez definido que el *wrapper* que se va a utilizar es el de *XLNets*, las posibles opciones a incluir son las siguientes:

- **xlnetCachingPeriod**: Propiedad que determina el periodo de tiempo, en segundos, durante el que se cachea la sesión de *XLNets*. Durante este periodo, el sistema no consulta los datos a *XLNets* para agilizar el rendimiento de la aplicación y no sobrecargar *XLNets*. Si el valor especificado es cero, la sesión de *XLNets* no se cacheará y cada vez que se haga una consulta se validarán las credenciales del usuario. En caso de no especificar la propiedad, el valor que tiene por defecto es 0, es decir, validación de cada petición.

- **specificCredentials**: Dentro de los posibles parámetros, que pueden o no ser especificados, dentro del *wrapper*, destaca la posibilidad de adecuar el objeto credenciales (*credentials*). Dicho objeto albergara toda la información, sobre el usuario (nombre, dni,…), de la que dispondrá el sistema de seguridad. La clase especificada para este parámetro, siempre, debe ser una extensión (*extend*) del objeto credenciales (*credentials*). El parámetro, en particular, debe especificar el *bean* que define la clase aplicada. Por defecto, el valor de esta parámetro será el objeto credenciales (*credentials*) original.

- **userChangeUrl**: Propiedad encargada de determinar la *url*, absoluta o relativa, a la que la aplicación redireccionará al navegador si éste detecta un cambio de usuario. Los cambios de

usuario durante una navegación no son habituales, aunque si posibles, y suelen responder a posibles ataques. No dispone de valor por defecto.

- **alternativeStorageUserCredentials**: Bajo ciertas circunstancias, es necesario trabajar con ciertos certificados almacenados en tarjeta, por ejemplo ***DNI's electrónicos***, de usuarios ajenos al personal de gobierno. Dichos usuarios responden al perfil de ciudadano anónimo que no se encuentra dado de alta en el LDAP de gobierno. Bajo estas circunstancias, puede que se dé la necesidad de tener algún usuario con un rol especial o un rol genérico para un grupo de usuarios. Para cubrir estas posibles necesidades de negocio, se ha incorporado la posibilidad de definir un *authentication provider* alternativo, sobre una base de datos, que permita almacenar esos datos especiales de seguridad. Mediante la propiedad "*alternativeStorageUserCredentials*" se especifica el *bean* que alberga la configuración de este sistema auxiliar de seguridad.

```
<!-- A conctinuación, se parametrizan los distintos aspectos asociada al manejo de
credenciales mediante una base de datos en Spring Security  -->
<bean id="jdbcUserService" class="com.ejie.x21a.security.SecurityCustomJdbcDaoImpl">
       <property name="dataSource" ref="dataSource"/>
       <property name="positionByUserdataQuery">
              <value> select PUESTO from CERT_USER_DATA where UPPER(USERNAME) = ? and
UPPER(DNI) = ? </value>
       </property>
       <property name="authoritiesByUserdataQuery">
              <value> select ROLES from CERT_USER_ROLES where UPPER(USERNAME) = ? and
UPPER(DNI) = ? </value>
       </property>
</bean>
```

El fragmento de código, se corresponde con la especificación de código asociada al almacenamiento alternativo de credenciales. El sistema de seguridad permite definir un *authentication provider* alternativo sobre base de datos. Ese *provider* alternativo, permitirá relacionar a los usuarios, autenticados mediante un certificado electrónico y que no tengan datos registrados en el LDAP de Gobierno, con su puesto y roles asociados.

El *bean* que define el *authentication provider* alternativo, tiene una serie de características y parámetros, específicos, que determinan su funcionamiento. Para poder usar correctamente dicho dispositivo es necesario identificar y comprender cada uno de los elementos:

  - **id**: Identifica el nombre del *bean* de definición. Este nombre es muy importante ya que debe corresponderse con el nombre especificado en la propiedad *alternativeStorageUserCredentials*.

  - **dataSource**: Entre los parámetros de configuración, requeridos, para el correcto funcionamiento del *provider* alternativo, destaca el *dataSource.* Dicho parámetro se encarga de gestionar el acceso a la base de datos donde se almacenan los datos. Normalmente, el *dataSource* utilizado será el propio de la base de datos de la aplicación (previamente definido) pero es posible definir otro a utilizar.

  - **class**: Determina la clase, propia de la aplicación, que implementa el acceso y gestión de los datos de la BBDD. Para que funcione correctamente el *provider* alternativo, es necesario que la clase utilizada extienda de la clase *UdaCustomJdbcDaoImpl.* Dicha clase contiene dos métodos abstractos (*loadUserPosition* y *loadUserAuthoritie*) que precisan ser implementados a la hora de crear la clase especificada en el *bean*.

```
public abstract class UdaCustomJdbcDaoImpl extends JdbcDaoImpl {

	private String positionByUserdataQuery = null;
	private String authoritiesByUserdataQuery = null;

	public UdaCustomJdbcDaoImpl(){
		super();
	}

	abstract protected String loadUserPosition(String userName, String dni,
N38API n38Api, Document xmlSesion);

	abstract protected Vector<String> loadUserAuthorities(String userName,
String dni, N38API n38Api, Document xmlSesion);


	//Getters & Setters

	protected String getPositionByUserdataQuery() {
		return this.positionByUserdataQuery;
	}

	public void setPositionByUserdataQuery(String positionByUserdataQuery)
	{
		this.positionByUserdataQuery = positionByUserdataQuery;
	}

	protected String getAuthoritiesByUserdataQuery() {
		return this.authoritiesByUserdataQuery;
	}

	public void setAuthoritiesByUserdataQuery(String
authoritiesByUserdataQuery)
	{
		this.authoritiesByUserdataQuery = authoritiesByUserdataQuery;
	}
}
```

Dichos métodos representan los dos accesos a base de datos necesarios para acceder a los datos que no poseen los usuarios autenticados a través de certificados, es decir, el puesto y los permisos (roles) del usuario.

- **positionByUserdataQuery**: Parámetro de configuración que determina la *query* utilizada a la hora de recuperar el puesto del usuario que realiza el acceso. Este parámetro tiene como objetivo, generalizar la creación de una clase, que pueda ser extendida entre aplicaciones o base de datos, adaptable a las exigencias de almacenamiento de cualquier esquema de datos. Un ejemplo de código extensible, en el que cambiando la *query* se podría usar en distintas aplicaciones o base de datos, sería el siguiente:

```
protected String loadUserPosition(String userName, String dni, N38API n38Api,
Document xmlSesion){

	List<String> result =
getJdbcTemplate().query(this.getPositionByUserdataQuery(), new String[] {
userName.toUpperCase(), dni.toUpperCase() }, new RowMapper<String>() {
		public String mapRow(ResultSet rs, int rowNum) throws
SQLException {
			return rs.getString(1);
		}
	});
```

```
        if(result.size() == 0){
                return(WrapperN38.getAnonymousProfile().get("position"));
        } else if(!(result.size() == 1)){
                IllegalArgumentException exc = new
IllegalArgumentException("loadUserPosition(): Revise su base de datos. El valor
de puesto, asociado a un usuario, debe ser único.");
                logger.error("loadUserPosition(): Revise su base de datos. El
valor de puesto, asociado a un usuario, debe ser único.", exc);
                throw exc;
        }

        return result.get(0);
```

- **authoritiesByUserdataQuery**: Parámetro de configuración que determina la *query* utilizada a la hora de recuperar los permisos asociados al usuario que realiza el acceso. Este parámetro tiene como objetivo, generalizar la creación de una clase, que pueda ser extendida entre aplicaciones o base de datos, adaptable a las exigencias de almacenamiento de cualquier esquema de datos. Un ejemplo de código extensible, en el que cambiando la *query* se podría usar en distintas aplicaciones o base de datos, seria el siguiente:

```
protected Vector<String> loadUserAuthorities(String userName, String dni,
N38API n38Api, Document xmlSesion){

        return new
Vector<String>(getJdbcTemplate().query(this.getAuthoritiesByUserdataQuery(),
new String[] { userName.toUpperCase(), dni.toUpperCase() }, new
RowMapper<String>() {
                public String mapRow(ResultSet rs, int rowNum) throws
SQLException {
                        return rs.getString(1);
                }
        }));
}
```

- **anonymousCredentials**: Propiedad múltiple, que se traduce en un *HashMap* en la clase, encargado de parametrizar ciertos datos de los usuarios que puedan no tener valor. Esta propiedad se usa, principalmente, para los casos en los que los usuarios usen certificados para identificarse y no dispongan datos registrados en el LDAP de Gobierno. También es posible utilizar este parámetro para almacenar datos de usuario anónimo que se puedan utilizar en alguna otra ubicación de la aplicación. Por defecto, los valores que se incluyen son:

  position => udaAnonymousPosition

  userProfiles => udaAnonymousProfile

- **alternativeOriginCredentialsApp**: Por diferentes cuestiones de negocio (por ejemplo, aplicaciones globales de gestión), es posible que ciertas aplicaciones requieran de los permisos de acceso de un usuario asociados aun código de aplicación diferente al propio. Por defecto, el *wrapper* de *XLNets* solo carga las instancias del usuario asociadas a la aplicación a la que esta accediendo. Si es preciso cargar las instancias del usuario, asociados a otro código de aplicación, es necesario utilizar este parámetro de configuración.

  Ante las posibles ocurrencias o requerimientos de las aplicaciones, el parámetro *alternativeOriginCredentialsApp*, puede aceptar diferentes formatos de entrada.

  Por un lado, se puede especificar, mediante un literal, el código de la aplicación cuyas instancias deban ser recuperadas. En este caso, el sistema de seguridad, además de recuperar de *XLNets*

las instancias del usuario asociadas al código de la aplicación especificado, recupera las instancias del código de la propia aplicación.

```
<property name="alternativeOriginCredentialsApp" value="y52b"></property>
```

Por otro lado, se puede especificar, mediante una lista de literales, un conjunto de códigos de aplicación cuyas instancias deban ser recuperadas. En este caso, el sistema de seguridad, además de recuperar de *XLNets* las instancias del usuario asociadas a cada uno de los códigos de aplicación especificados, recupera las instancias del código de la propia aplicación.

```
<property name="alternativeOriginCredentialsApp">
	<list>
		<value>y52b</value>
		<value>y68b</value>
		<value>o75b</value>
	</list>
</property>
```

Finalmente, se puede especificar, un *bean* que defina la clase encargada de devolver el código o códigos de aplicación cuyas instancias deban ser recuperadas. Esta clase, debe implementar la clase *AlternativeOriginCredentialsApp*. En este caso, el sistema de seguridad, además de recuperar de *XLNets* las instancias del usuario asociadas a la lista de códigos de aplicación que devuelve el método *getAppCodes* de la clase, recupera las instancias del código de la propia aplicación.

```
<bean id="perimetralSecurityWrapper"
class="com.ejie.x38.security.PerimetralSecurityWrapperN38Impl">

	.
	.
	.

	<property name="alternativeOriginCredentialsApp" ref="aplicacionesAceptadas"/>

	.
	.
	.

</bean>

<!-- Se especifica un filtro de exclusión sobre los usuarios validados. Esto permite
la limitación del acceso aunque se han usuarios validos de XLNets. -->

	<bean id="aplicacionesAceptadas"
class="com.ejie.x21a.security.ObtencionAplicaciones"/>
```

- **excludeFilter**: Bajo ciertas circunstancias o exigencias de negocio, es posible que se requiera de un filtro adicional para permitir o no el acceso de cierto tipo o familia de usuarios a las aplicaciones. Si se desea limitar el acceso de usuarios que no utilicen certificados o se desea excluir un tipo de familia o colectivo de usuarios, es posible diseñar un filtro, adicional al sistema de seguridad, que permita validar los usuarios que desean acceder. Aunque, según la política de gestión del *authentication provider*, ciertos usuarios estarían autorizados a acceder, si el filtro de exclusión no da su visto bueno las credenciales del usuario serán rechazadas y no podrá acceder a la aplicación. Mediante la propiedad "*excludeFilter*" se especifica el *bean* que alberga la configuración del filtro de exclusión. Todos los aspectos relacionados con la configuración de este servicio, se explican en detalle un poco más adelante.

4.2.2. Wrapper del Mock de seguridad

A diferencia de *XLNets*, el *mock* no es un *authentication provider* como tal. El *mock* no es más que un módulo que pretende simular el comportamiento de *XLNets* para las fases de desarrollo.

Por diferentes cuestiones (trabajar en una oficina sin conexión, no disponer de los permisos de usuario, esperas por gestiones burocráticas,…), es posible que no se disponga de la infraestructura o la configuración apropiada de *XLNets*. Bajo estas circunstancias, la gestión de la seguridad, y el diseño de la misma quedaría paralizada a la espera de disponer de una plataforma de desarrollo propicia. Este hecho puede provocar penalizaciones en el desarrollo y en los tiempos de finalización. Para evitar este tipo de problemas y sus derivados, nace el *mock*.

El principal objetivo del *mock* es proporcionar una herramienta de simulación de usuarios de *XLNets*, que permita trabajar en la seguridad de la aplicación, sin necesidad de disponer del propio *XLNets*. El *wrapper* del *mock*, ha sido diseñado para permitir definir todos los datos necesarios en la aplicación que se recogen de *XLNets* (nombre de usuario, permisos, nombre de pila,…) y utilizarlos como si de un usuario autenticado se tratase.

De todo el desarrollo del *wrapper* han surgido una serie de parámetros destinados a cubrir todas las posibilidades funcionales del *mock* (tanto desde el punto de vista funcional del *mock* como desde el punto de vista de *Spring* y de *Spring Security*). Los distintos parámetros que se pueden, o deben, utilizar son los siguientes:

```
<bean id="perimetralSecurityWrapper"
class="com.ejie.x38.security.PerimetralSecurityWrapperMockImpl">
	<property name="principal">
		<list>
			<map>
				<entry key="userName" value="udaAdministrador"/>
				<entry key="name" value="Uda"/>
				<entry key="surname" value="Administrador"/>
				<entry key="fullName" value="Uda Administrador"/>
				<entry key="nif" value="12121212j"/>
				<entry key="policy" value="1"/>
				<entry key="position" value="01"/>
				<entry key="isCertificate" value="no"/>
				<entry key="roles">
					<list>
						<value>ADMIN_UDA</value>
						<value>X21A-IN-0003</value>
					</list>
				</entry>
			</map>
			<map>
				<entry key="userName" value="udaUser"/>
				<entry key="name" value="Uda"/>
				<entry key="surname" value="User"/>
				<entry key="fullName" value="Uda User"/>
				<entry key="nif" value="17398234h"/>
				<entry key="policy" value="1"/>
				<entry key="position" value="02"/>
				<entry key="isCertificate" value="si"/>
				<entry key="roles">
					<list>
						<value>USER_UDA</value>
						<value>X21A-IN-0003</value>
					</list>
				</entry>
```

```
                                <entry key="subjectCert">
                                        <map>
                                                <entry key="OID.1.3.6.1.4.1.18838.1.1"
value="11111111H"/>
                                                <entry key="SERIALNUMBER" value="A01017169"/>
                                                <entry key="surname" value="Uda"/>
                                                <entry key="GIVENNAME" value="Empresa"/>
                                                <entry key="CN" value="Uda Empresa"/>
                                                <entry key="DNQ" value="-cif A06617169"/>
                                                <entry key="OU" value="Condiciones de uso en
www.izenpe.com nola erabili jakiteko"/>
                                                <entry key="OU" value="Entitatearen ziurtagiria
- Certificado de entidad"/>
                                                <entry key="OU" value="Ziurtagiri onartua -
Certificado reconocido"/>
                                                <entry key="O" value="Uda Empresa"/>
                                                <entry key="C" value="ES"/>
                                        </map>
                                </entry>
                        </map>
                </list>
        </property>

        <property name="userChangeUrl"
value="http://desarrollo.jakina.ejiedes.net:7001/x21aMantenimientosWar/"/>

        <property name="specificCredentials" ref="aplicationSpecificCredentials"/>
</bean>


<bean id="aplicationSpecificCredentials" class="com.ejie.x21a.security.myUserCredentials"/>
```

El *mock*, a nivel general de parámetros, solo dispone de dos:

- **principal**: Es un parámetro *list*, que alberga objetos *map*, en el que se especifican los distintos usuarios con sus datos asociados. Cada uno de los *maps* especificados almacena todos los datos asociados a un usuario:
    - **username**: Corresponde con el identificador de usuario.
    - **name**: Nombre de pila del usuario.
    - **surname**: Apellidos del usuario
    - **fullname**: Nombre completo del usuario. Valor utilizado para presentar, en pantalla, el usuario que esta autenticado.
    - **nif**: Nif del usuario virtual.
    - **policy**: Política aplicada al usuario. Este valor tiene sentido cuando se trabaja con certificados ya que identifica el tipo de certificado utilizado (ciudadano, funcionario,…). Carece de sentido para los usuarios que utilicen la autenticación de usuario y *password* (*XLNets* devuelve datos virtuales para esos casos).
    - **position**: Puesto que ocuparía el usuario.
    - **isCertificate**: Determina si el usuario utiliza un certificado, o no, para autenticarse.
    - **roles**: Permisos de los que dispone el usuario. Como en el caso de *XLNets*, el sistema añadirá la cadena "*role_*" a los valores indicados.
    - **subjectCert**: Conjunto de información almacenada en el certificado. El objetivo de este conjunto de datos es simular la información que seria aportada por un certificado. Es importante tener en cuenta que, el campo *isCertificate* debe tener el valor "si" para que el sistema cargue los datos asociados al certificado. Para poder componer una

simulación apropiada, es esencial especificar una información similar a la que aportarían los certificados reales.

Por defecto, aunque no se defina ningún usuario, el *mock* siempre dispone de un usuario anónimo para acceder al sistema. Este usuario, siempre está disponible y definido con datos genéricos (sin permisos de acceso).

- **userChangeUrl**: Propiedad encargada de determinar la *url*, absoluta o relativa, a la que la aplicación redirecciona al navegador si esta detecta un cambio de usuario. No dispone de valor por defecto.

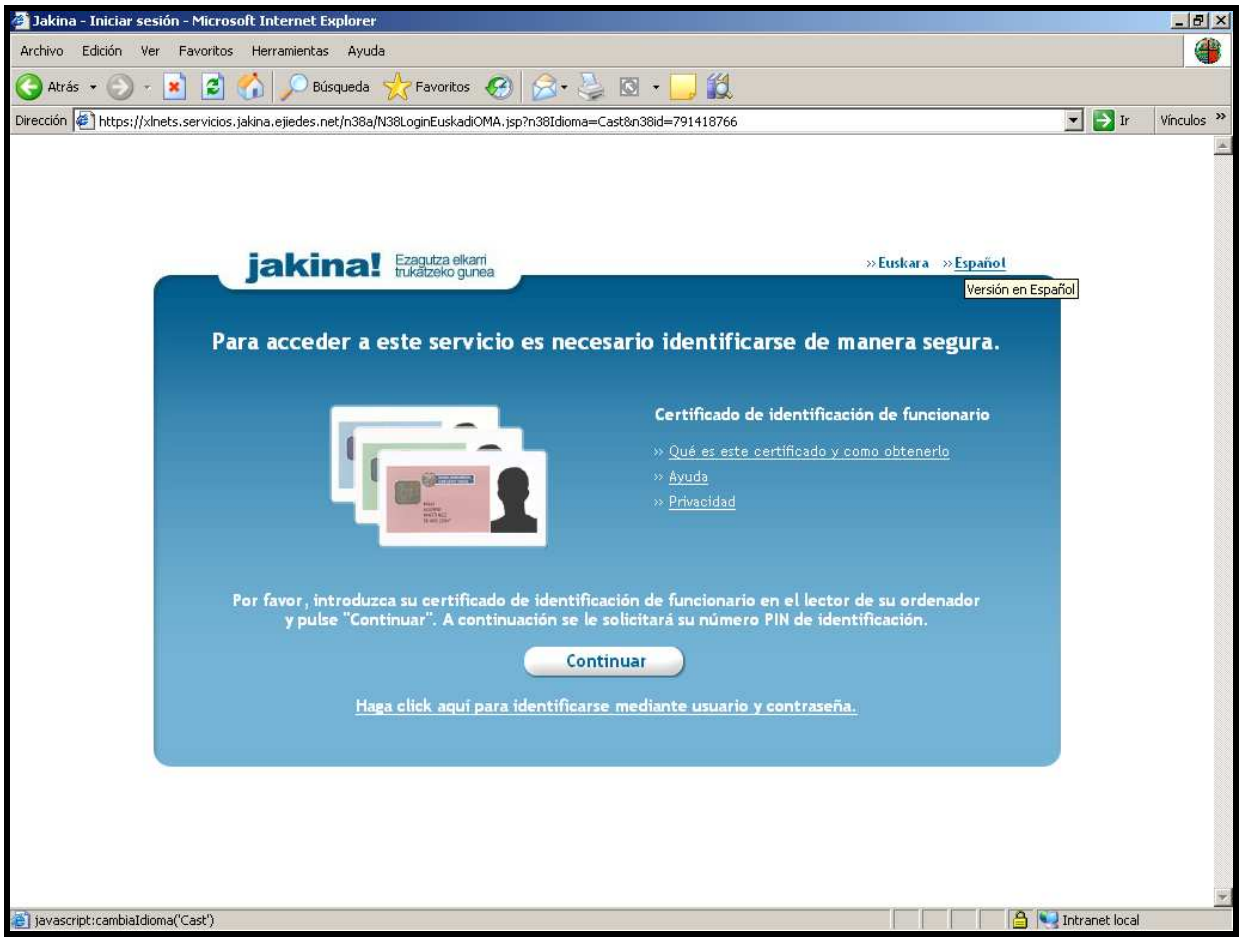
- **specificCredentials**: Dentro de los posibles parámetros, que pueden o no ser especificados, dentro del *wrapper*, destaca la posibilidad de adecuar el objeto credenciales (*credentials*). Dicho objeto albergara toda la información, sobre el usuario (nombre, dni,…), de la que dispondrá el sistema de seguridad. La clase especificada para este parámetro, siempre, debe ser una extensión (*extend*) del objeto credenciales (*credentials*). El parámetro, en particular, debe especificar el *bean* que define la clase aplicada. Por defecto, el valor de esta parámetro será el objeto credenciales (*credentials*) original.

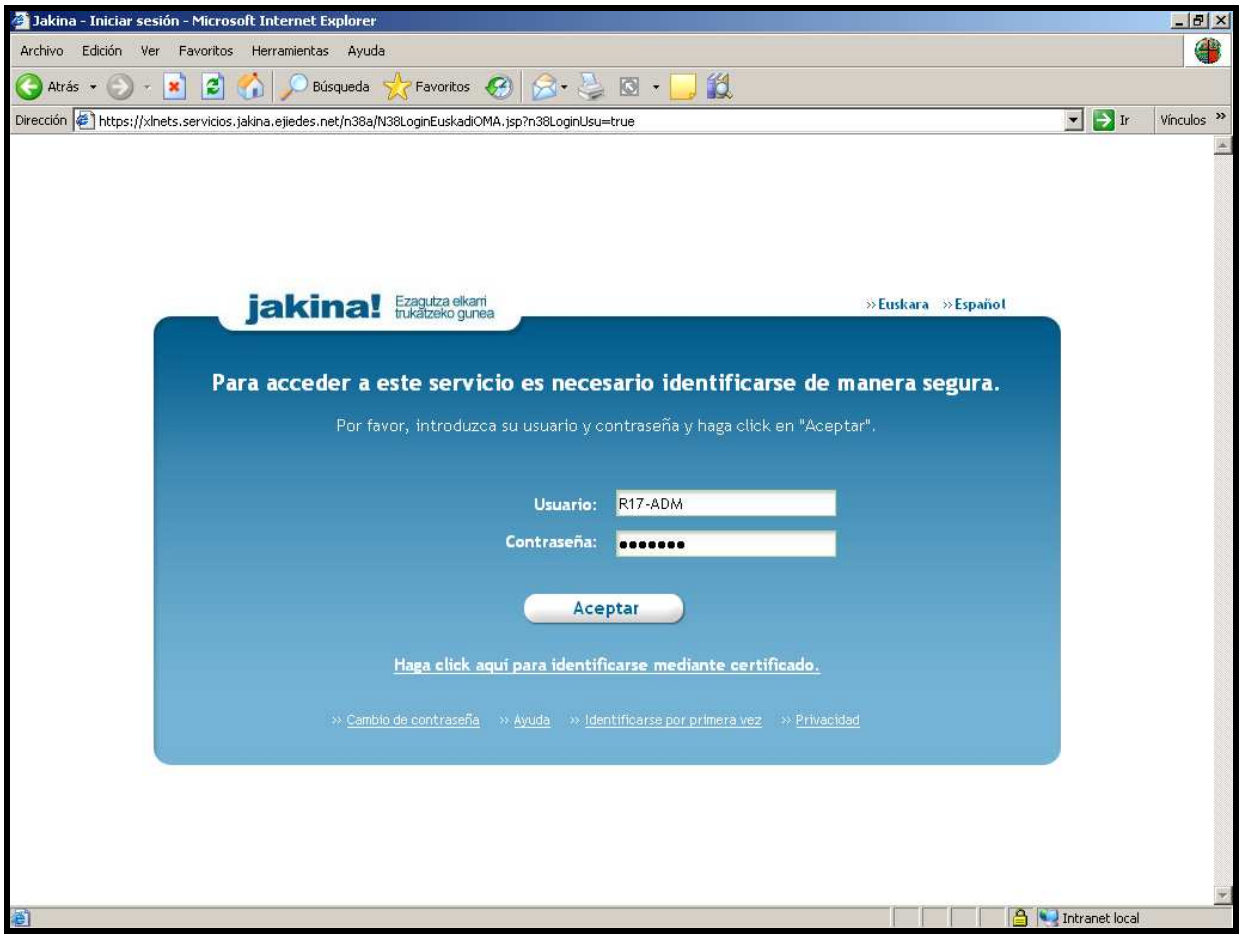
5. Funcionamiento conceptual

Cuando se realiza un acceso a un recurso protegido de una aplicación desarrollada con **UDA**, según este configurado el *wrapper* de *XLNets* o el *wrapper* del *mock*, el usuario tendrá que interaccionar con las pantallas asociadas al sistema de autenticación. En caso de no autenticarse correctamente o no ser capaz de hacerlo, el acceso al recurso solicitado se le será negado.

5.1. Acceso mediante *XLNets*

1. La petición entra por la cadena de filtros definida en *Spring Security* detectando si se ha realizado una petición *http* a un recurso securizado y comprobando en caso afirmativo si existe o no un usuario autenticado válido.
2. Si no existe usuario autenticado, *Spring Security* delega la autenticación en el módulo de *login* proporcionado por *XLNets*.





3. Una vez autenticado, *Spring Security* inserta en sesión las credenciales y autorizaciones del usuario, comprobando si dentro de las autorizaciones existe una que le permita acceder al recurso solicitado.
4. En caso de que no tener permisos adecuados, se redirige a la página de acceso denegado. Dicha pagina viene dada por la configuración base definida en el fichero "*security-core-config.xml*". Por defecto, su valor es "/accessDenied" y su aspecto es:

La pagina de acceso denegado, por defecto, es muy básica. Es recomendable que las aplicaciones adapten dicha página a los estilos y exigencias de su negocio.

5. En caso de tener permisos, se accederá al recurso solicitado normalmente.

5.2. Acceso mediante el *mock*

1. La petición entra por la cadena de filtros definida en *Spring Security* detectando si se ha realizado una petición *http* a un recurso securizado y comprobando en caso afirmativo si existe o no un usuario autenticado válido.

2. Si no existe usuario autenticado, *Spring Security* delega la autenticación en el módulo de *login* proporcionado por el *mock*.

3. La pagina de autenticación del *mock*, al seleccionar un usuario e iniciar sesión, añadirá la *cookie* "*udaMockUserName*" con el nombre del usuario autenticado para el sistema de seguridad.

4. Una vez autenticado, *Spring Security* inserta en sesión las credenciales y autorizaciones del usuario, comprobando si dentro de las autorizaciones existe una que le permita acceder al recurso solicitado.

5. En caso de que no tener permisos adecuados, se redirige a la página de acceso denegado. Dicha pagina viene dada por la configuración base definida en el fichero "*security-core-config.xml*". Por defecto, su valor es "/accessDenied" y su aspecto es:

La pagina de acceso denegado, por defecto, es muy básica. Es recomendable que las aplicaciones adapten dicha página a los estilos y exigencias de su negocio.

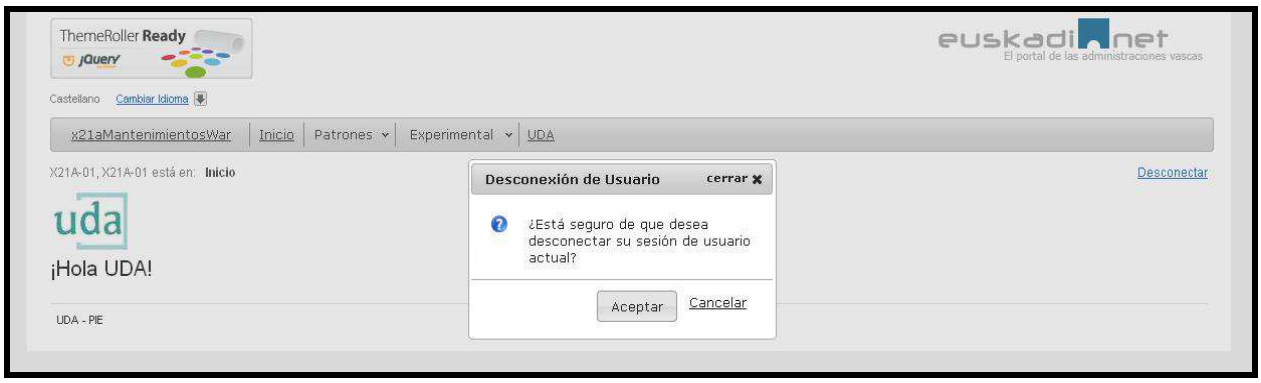
6. En caso de tener permisos, se accederá al recurso solicitado normalmente.

5.3. Desconexión de usuario

Entre los distintos aspectos que pueden intervenir en el desarrollo de una aplicación, uno de los temas más críticos e importantes es la gestión de la seguridad. Dentro de los distintos elementos de interacción con el modulo de seguridad, destacan la necesidad de trasmitir, en todo momento, el estado del usuario (autenticado o no) y la disposición de un sistema para la desconexión del sistema.

Para facilitar, tanto, la gestión de la desconexión de los usuarios, como, la especificación del usuario autenticado, el componente de migas incluye la capacidad de presentar el nombre completo del usuario autenticado y un enlace para permitir la desconexión, bajo demanda, del sistema de seguridad.

El procedimiento de desconexión, automáticamente, presenta un mensaje de confirmación para completar el proceso.

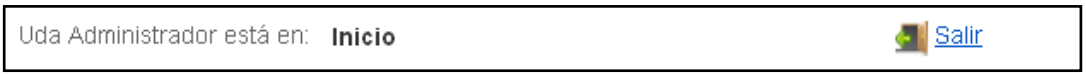


Además de permitir a los usuarios interaccionar con su información de seguridad, el componente de migas también exterioriza el concepto de desconexión o salida manejado por el sistema de seguridad. Según la especificación de parámetros realizada, el sistema de seguridad permite a los usuarios que salgan, únicamente, de la aplicación o que, además, finalicen la sesión del *authentication provider* (terminando con ellos su sesión de usuario).

Según el valor especificado en la variable *destroySessionSecuritySystem* de las credenciales (*credentials*), solo aplicable a la definición del *wrapper* de *XLNets* (para el *wrapper* del *mock* siempre tendrá el valor *false*), el componente presentara uno de los siguientes aspectos:

Con desconexión del *authentication provider*:

Sin desconexión del *authentication provider*:



El valor de la variable *destroySessionSecuritySystem* de las credenciales (*credentials*) es fijado por el valor, asociado al *bean myLogoutHandler*, de la propiedad *invalidateUserSession*. Dicho *bean* viene especificado en el fichero de configuración "*security-core-config.xml*" y corresponde con el módulo de desconexión (*logOut*) del sistema de seguridad.

```
<!-- Para la completa interacción con XLNets se especifica un "logoutHandler" especifico -->
<bean id="myLogoutHandler" class="com.ejie.x38.security.MyLogoutHandler">
       <property name="perimetralSecurityWrapper" ref="perimetralSecurityWrapper" />
       <property name="invalidateHttpSession" value="true" />
       <property name="invalidateUserSession" value="true" />
</bean>
```

6. Acceso y gestión de datos de seguridad

Una vez se ha realizado la correcta autenticación de los usuarios, el sistema de seguridad (*Spring Security*) crea y almacena, en sesión, un objeto "*authentication*" con los diferentes datos asociados a la sesión del usuario. Entre los distintos datos u objetos que almacena, destaca el objeto credenciales (*credentials*) que se encarga de recopilar todos los datos extraídos del *authentication provider* sobre el usuario.

La clase que implementa el objeto credenciales (*credentials*) recopila los siguientes datos:

```
package com.ejie.x38.security;

import java.io.Serializable;
.
.
.
/**
 *
 * @author UDA
 *
 */
public class UserCredentials implements Serializable {

	private static final long serialVersionUID = 1L;

	private String nif;
	private String policy;
	private String userName;
	private String name;
	private String surname;
	private String fullName;
	private String position;
	private String uidSession;
	private boolean isCertificate;
	private String udaValidateSessionId;
	private Vector<String> userProfiles;
	private boolean destroySessionSecuritySystem = false;
	private DynaBean subjectCert = null;
	private ArrayList<String> userDataProperties;

	//Constructor, Getters & Setters
	.
	.
	.
	//Functions of Data gestion
	public String toString() ;

	public void loadCredentialsData(PerimetralSecurityWrapper perimetralSecurityWrapper,
HttpServletRequest request);

	//Adaptation method for applications
	protected void afterCredentialsCreation(PerimetralSecurityWrapper
perimetralSecurityWrapper, HttpServletRequest request);

	//Functions to manage the SubjectCert's data
	public boolean containsSubjectCert(String id);

	public String getSubjectCert(String data);

	public void setSubjectCert(String property, String value){

	public void deleteSubjectCert(String property){
}
```

- **nif**: Nif del usuario autenticado.
- **policy**: Política aplicada al usuario. Este valor tiene sentido cuando se trabaja con certificados ya que identifica el tipo de certificado utilizado (ciudadano, funcionario,…). Carece de sentido para los usuarios que utilicen la autenticación de usuario y *password* (*XLNets* devuelve datos virtuales para esos casos).
- **username**: Corresponde con el identificador de usuario.
- **name**: Nombre de pila del usuario.
- **surname**: Apellidos del usuario
- **fullname**: Nombre completo del usuario. Valor utilizado para presentar, en pantalla, el usuario que está autenticado.
- **position**: Puesto ocupado por el usuario.
- **uidSession**: Identificador de la sesión de seguridad abierta en el *authentication provider*. En el caso de *XLNets*, se extrae el valor del propio *provider*, en el caso del *mock*, se utiliza el identificador de la sesión de usuario de *Weblogic*.
- **isCertificate**: Determina si el usuario ha utilizado un certificado, o no, para autenticarse.
- **udaValidateSessionId**: Identificador, inequívoco, de la autenticación del usuario en el sistema. Dicho valor sirve para identificar cambios de usuario, pérdidas o cierres de la sesión del *authentication provider* y para identificar intentos de usurpar identidad.
- **userProfiles**: Permisos de los que dispone el usuario.
- **destroySessionSecuritySystem**: Parámetro, solo aplicable a la definición del *wrapper* de *XLNets* (para el *wrapper* del *mock* siempre tendrá el valor *false*), que determina el cierra de la sesión en el *authentication provider*. Por defecto, su valor siempre es *false*.
- **subjectCert**: Objeto dinámico, capaz de almacenar la información (dependiente del certificado) proporcionada por los distintos certificados. La variable, como tal, no es accesible de forma directa. Para su manipulación, es necesario utilizar los métodos *containsSubjectCert*, *getSubjectCert*, *setSubjectCert* y *deleteSubjectCert*. En caso de no realizarse una autenticación mediante certificado, el objeto siempre estará vacio (*null*)
- **userDataProperties**: Listado de literales, encargado de almacenar el nombre de los distintos campos recopilados del certificado. El objetivo de este campo es registrar y proporcionar los distintos nombres de los datos extraídos del certificado. En caso de precisar los literales de los distintos campos obtenidos, es posible acceder al objeto mediante su función *get* asociada o rastrear un literal determinado mediante el método *containsSubjectCert*.

Todos estos datos del usuario, están gestionados por el sistema de seguridad pero no solo son accesibles por él. La lógica de negocio de cualquier aplicación (para acceder a servicios especiales, para identificar acciones del usuario,…), es posible que requiera de los datos obtenidos por el sistema de seguridad. Aunque, siempre es posible pedir de nuevo los datos al *authentication provider*, carece de sentido volver a pedir datos de los que ya se dispone.

El acceso de las credenciales de usuario se puede realizar de tres formas diferentes:

1. Siempre que se disponga del objeto *request* se puede obtener de la sesión el objeto "*SPRING_SECURITY_CONTEXT*" que alberga el objeto "*authentication*". En las *jsps* de la aplicación, como se dispone de acceso a la *request* y a la sesión de usuario, se puede obtener el objeto credenciales (*credentials*).

```
<script type="text/javascript">
  Var CREDENCIALES = "${sessionScope.SPRING_SECURITY_CONTEXT.authentication.credentials}";
</script>
```

2. Es posible acceder, a ciertos datos del contexto de seguridad, mediante los *tags* de *Spring Security* para las *jsps*. El *tag* "*Authentication*" es capaz de renderizar una propiedad del contexto de seguridad, directamente, sobre la *jsp*.

```
<sec:authentication property="principal" /><br/>
<sec:authentication property="credentials" /><br/>
<sec:authentication property="credentials.fullName" /><br/>
```

3. Siempre que se este trabajando en una clase incluida en el contexto de *spring*, es posible invocar al objeto "*SecurityContextHolder*" para obtener los objetos de seguridad.

```
import com.ejie.x38.security.UserCredentials;

UserCredentials userCredentials = null;

userCredentials = (UserCredentials)
SecurityContextHolder.getContext().getAuthentication().getCredentials();
```

4. Para facilitar y mejorar el uso, sincronizado, de los datos asociados a las credenciales en las *jsps*, es posible hacer accesible, los mismos, a partir del contexto de *Spring*. Para poder disponer, de esta manera, de los datos de las credenciales, y evitar con ello los posibles problemas de sincronización asociados al almacenaje en sesión, es necesario facilitar a la capa de presentación el acceso al *bean udaAuthenticationProvider* alojado en el contexto de *Spring*. Para ello, se debe especificar, en el fichero de configuración "*mvc-config.xml*", la exposición del *bean* en el objeto *viewResolver*.

```
<bean id="viewResolver" class="com.ejie.x38.control.view.UdaViewResolver">
	<property name="viewClass" value="com.ejie.x38.control.view.UdaTilesView"/>
	<property name="exposedContextBeanNames" >
		<list>
			<value>localeResolver</value>
			<value>mvcInterceptor</value>
			<value>udaAuthenticationProvider</value>
		</list>
	</property>
</bean>
```

Una vez declarado el *bean*, es posible acceder, en las *jsps*, a los datos del mismo mediante Expression Language (EL), jstl,…

```
<script type="text/javascript">
	Var CREDENCIALES = "${udaAuthenticationProvider.userCredentials}";
</script>
```

6.1. extensibilidad del objeto credenciales

Además de los distintos campos y sus respectivas funciones de gestión, el objeto credenciales (*credentials*) dispone de una función de gestión denominada *afterCredentialsCreation*. Dicha función

tiene como objetivo proporcionar un punto de extensibilidad cómodo para hacer pequeños ajustes a la hora de gestionar las credenciales. El funcionamiento general de carga de las credenciales incluye la ejecución de dicho método en último lugar. Esto unido a la posibilidad de definir un objeto específico de credenciales en el *wrapper* de *XLNets*, permite definir, en el método, un proceso específico que ajuste la creación de credenciales a las necesidades de una aplicación en particular (alterar o ajustar datos almacenados, la carga de nuevos campos,…).

El uso de este punto de extensibilidad es, relativamente, sencillo. En primer lugar se debería extender la clase credenciales mediante una clase propia de la aplicación. Dicha clase entendedora, deberá incluir la sobre escritura del método *afterCredentialsCreation*, con el tratamiento especifico requerido por la aplicación, y el resto de elementos (nuevas funciones, nuevos campos,…) precisados.

```
package com.ejie.x21a.security;

public class myUserCredentials extends UserCredentials {

	private static final long serialVersionUID = 1L;

	@Override
	public void afterCredentialsCreation(PerimetralSecurityWrapper
perimetralSecurityWrapper, HttpServletRequest request){

		/* Código especifico de la aplicación */

	}
}
```

Una vez creada la extensión del objeto credenciales (*credentials*), mediante la propiedad "*specificCredentials*", se especificaría su uso en la definición del propio *wrapper* (extensibilidad dinámica esta presente en cualquiera de los *wrappers* disponibles).

```
<bean id="perimetralSecurityWrapper"
class="com.ejie.x38.security.PerimetralSecurityWrapperN38Impl">

	.
	.
	.

      <property name="specificCredentials" value="com.ejie.x21a.security.MyUserCredentials"/>

	.
	.
	.

</bean>
```

Este sería el procedimiento más sencillo para ajustar la creación de las credenciales (*credentials*) en el sistema de seguridad. La infraestructura permite el cambio total del objeto credenciales (*credentials*), ya que se puede especificar el objeto que se usará para ello. A pesar de que existe la posibilidad, esta práctica no es del todo recomendable. La modificación de los distintos métodos de creación, carga y gestión de datos de las credenciales (*credentials*), podría impactar en el propio sistema de seguridad y la responsabilidad de la codificación de los cambios necesarios tendrá que ser mantenida y soportada por la propia aplicación.