Capítulo 8

Detalles técnicos del sistema implementado

Contenido

8.1 Introducción

En este capítulo se describirán más detenidamente algunos aspectos prácticos del sistema implementado que, por razones de claridad, no se incluyeron en el anterior. Este Proyecto se ha realizado usando Eclipse pero la información contenida en este capítulo

es válida también para cualquier otro entorno de trabajo. Las instrucciones específicas para Eclipse se recogen en el capítulo 9.

El sistema requiere el contenedor de aplicaciones Tomcat 5.5 y, como soporte del código Java, la versión 5.0 del J2SE. Es posible hacerlo funcionar con versiones anteriores, tanto de Java como de Tomcat, pero en esta memoria no se contempla esta posibilidad por ser más complicada (se remite al lector a la bibliografía si quiere explorar esa opción).

Con respecto a los archivos que componen la implementación, existen dos carpetas principales, conteniendo cada una un proyecto Java independiente. Los nombres de las carpetas son los siguientes:

- **shib**

  Aquí se recoge todo el código Java y los archivos de configuración correspondientes a las aplicaciones web "shibboleth-idp" y "shibboleth-sp".

- **shib-filter**

  En este directorio está el código de la aplicación "secure", el del filtro de acceso, sus archivos de configuración correspondientes y aquellos recursos a los que el usuario desea acceder: historial.htm y documento_no_protegido.htm).

8.2 Compilación del código

Los dos proyectos tienen procesos de compilación autónomos. En realidad, llevaremos a cabo un proceso que no sólo compilará el código, sino que, como veremos a continuación, realizará todas las operaciones necesarias para preparar el despliegue de las aplicaciones web, empaquetando las clases compiladas, creando los archivos WAR, copiando estos archivos en los directorios adecuados en la estructura de Tomcat, etc. Para llevar a cabo todas estas operaciones, hacemos uso de **Ant**, una utilidad escrita en Java, incluida por defecto en Eclipse. Ant no es parte de la distribución Java de Sun, sino que proviene de la Apache Software Foundation.

Ant toma como entrada un archivo en formato XML, **build.xml**, en el cual se especifican uno o más ***targets***. Cada target consiste en una serie de operaciones para crear directorios, copiar ficheros, compilar código, etc. En el archivo build.xml algunos nombres de carpetas y ficheros se especifican como variables. El valor de dichas variables puede asignarse dentro del mismo archivo build.xml, leerse de otro archivo llamado **build.properties** o establecerse usando la interfaz de Eclipse.

Veamos a continuación, por separado, los procesos de compilación y "construcción" de los dos proyectos Java.

Compilación y construcción del proyecto Java "shib"

Dentro de las carpetas de este proyecto se incluye un archivo build.xml que Ant tomará como entrada para el proceso de construcción de la aplicación. En este caso, las variables, correspondientes a nombres de archivos y directorios, se leerán de otro archivo incluido en el proyecto, llamado build.properties. Las líneas de dicho archivo que nos interesan aparecen en la figura 31.

```
tomcat.home=C:/Archivos de programa/eclipse/jakarta-tomcat-5.5.9

idp.webapp.name=shibboleth-idp
sp.webapp.name=shibboleth-sp

idp.deployment.descriptor=dist.idp-container-security-example.xml
sp.deployment.descriptor=dist.sp.xml

idp.home=/usr/local/shibboleth-idp
sp.home=/usr/local/shibboleth-sp
```

Figura 31: Fragmento del archivo build.properties

Los parámetros de la figura 31 tienen el siguiente significado:

`tomcat.home = C:/Archivos de programa/eclipse/jakarta-tomcat-5.5.9`

Proporciona el nombre completo de la ruta donde Tomcat está instalado.

idp.webapp.name = shibboleth-idp

El nombre del archivo WAR que Ant generará para la aplicación web correspondiente al Proveedor de Identidad.

sp.webapp.name = shibboleth-sp

El nombre del archivo WAR que Ant generará para la aplicación web encargada del Assertion Consumer Service y del Attribute Requester.

idp.deployment.descriptor = dist.idp-container-security-example.xml

El nombre del archivo que se usará como descriptor de despliegue para la aplicación web del IdP. Dicho archivo se toma del directorio shib\WebAppConfig y se empaquetará dentro del fichero WAR de la aplicación web, en nuestro caso shibboleth-idp.war.

sp.deployment.descriptor = dist.sp.xml

El nombre del archivo que se usará como descriptor de despliegue para la aplicación web designada por el parámetro `sp.webapp.name`. Dicho archivo se toma del directorio shib\WebAppConfig y se empaquetará dentro del fichero WAR de la aplicación web, en nuestro caso shibboleth-sp.war.

idp.home=/usr/local/shibboleth-idp

El sistema requiere de la existencia de un directorio donde almacenar la configuración del Proveedor de Identidad. En este caso, y si la unidad de disco en la que trabajamos tiene la letra C como distintivo, el directorio se crearía en c:\usr\local\shibboleth-idp. A pesar que el nombre elegido en este caso para la carpeta tiene "estilo Unix", puede utilizarse también en sistemas Windows (como se ha hecho en este Proyecto) sin que exista ningún problema. Hay más información sobre el contenido de los directorios de usr\local en los apartados 8.6, 8.7 y 9.4.

sp.home=/usr/local/shibboleth-sp

Igual que en el apartado anterior pero, en este caso, para la aplicación web del Proveedor de Servicios.

Lanzaremos Ant para que ejecute las órdenes contenidas en el archivo build.xml de la carpeta del proyecto Java Shib. En concreto, especificaremos que lleve a cabo las operaciones contenidas en los siguientes targets (y en el orden en que se enumeran aquí):

- **target "clean-install"**
  Borra los directorios de configuración de las aplicaciones web shibboleth-idp y shibboleth-sp, es decir c:\usr\local\shibboleth-idp y c:\usr\local\shibboleth-sp así como las carpetas correspondientes a estas dos aplicaciones que se encuentran dentro del directorio "webapps" de Tomcat.
- **target "compile"**
  Compila el código Java.
- **targets "install.idp.filesystem" e "install.sp.filesystem"**
  Crean nuevos archivos WAR, los copian a la carpeta "webapps" de Tomcat y crean nuevos directorios para shibboleth-idp y shibboleth-sp.
- **target "clean-all"**
  Borra las clases compiladas y las copias de los archivos WAR que han quedado dentro de la carpeta "Shib" tras los pasos anteriores (y que ya no tienen función alguna). No borra nada en c:\usr\local ni en "webapps".

Compilación y construcción del proyecto Java "shib-filter"

Se compilan y preparan para su despliegue el filtro de acceso y la aplicación web "secure". En este caso los parámetros para la construcción están contenidos en el propio archivo build.xml. En ese archivo hay que prestar atención a la siguiente línea, que indica donde hemos instalado el contenedor de aplicaciones:

```
<property name="tomcat.home"
value="C:\Archivos de programa\eclipse\jakarta-tomcat-5.5.9" />
```

Ejecutamos los siguientes “targets” usando Ant:

- **target “build”**
  Compila el código correspondiente al filtro y lo empaqueta en el archivo shib-filter.jar.
- **target “deploy”**
  Copia el mencionado archivo shib-filter.jar en el directorio de Tomcat \shared\lib.
- **target “deploy-testapp”**
  Crea el archivo “secure.war” y lo copia a la carpeta “webapps” de Tomcat.

8.3 Configurar los servidores

Configurar el nombre en Windows

Para simular la existencia de dos servidores utilizando sólo el “localhost”, modificaremos el archivo de “hosts” de Windows, cuya ruta completa es la siguiente:

```
{windows}\system32\drivers\etc\hosts
```

donde `{windows}` representa el directorio raíz de la instalación de Windows.
Añadiremos las dos líneas que aparecen a continuación:

```
127.0.0.1 idp.example.org
127.0.0.1 sp.example.org
```

Si el archivo no existe, lo crearemos con esas dos líneas. Por supuesto, esto es sólo para un entorno experimental. En producción cada servidor debería tener su dirección IP real.

Asociar servidores y puertos con certificados en Tomcat

Para que los dos servidores puedan comunicarse usando comunicación segura, modificaremos el archivo siguiente:

`{tomcat.home}\conf\server.xml`

`{tomcat.home}` representa aquí la carpeta principal raíz donde hemos instalado el contenedor de aplicaciones. Añadiremos un bloque de texto donde se asociarán dos archivos de claves de Java a unos puertos determinados de Tomcat. El texto se muestra en la figura 32.

```
<Connector port="443" maxHttpHeaderSize="8192"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true"
acceptCount="100" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS"
keystoreFile="conf/idp-example.jks" keystorePass="exampleorg"
/>
 <Connector port="8443" maxHttpHeaderSize="8192"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true"
acceptCount="100" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS"
keystoreFile="conf/idp-example.jks" keystorePass="exampleorg"
/>
<Connector port="9443" maxHttpHeaderSize="8192"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true"
acceptCount="100" scheme="https" secure="true"
clientAuth="false" sslProtocol="TLS"
keystoreFile="conf/sp-example.jks" keystorePass="exampleorg"
/>
```

Figura 32: Configuración de SSL en Tomcat

8.4 Configurar usuarios y contraseñas

Cuando se realiza una llamada al Servicio Single Sign-On del Proveedor de Identidad queremos que se active la Autenticación Básica de Tomcat para que se fuerce al navegador a pedir al usuario que introduzca un nombre y una contraseña. Para ello debemos modificar dos archivos. El primero de ellos es un fichero de configuración de Tomcat cuyo nombre completo es :

```
{tomcat.home}/conf/tomcat-users.xml
```

Cada línea de ese archivo representa un nombre de usuario y una contraseña. En nuestro caso, la línea que nos interesa es la siguiente:

```
<user username="tomcat" password="tomcat" roles="tomcat"/>
```

Como se puede ver, es posible asignar un determinado rol o conjunto de roles a un usuario aunque para nosotros esa posibilidad no tendrá utilidad, ya que para todo lo relacionado con roles y credenciales disponemos de SAML.

El segundo archivo a modificar es el descriptor de despliegue de la aplicación web de la que el Single Sign-On Service forma parte, esto es, shibboleth-idp. El archivo se encuentra dentro de la carpeta del proyecto Java “shib”, en concreto en esta ubicación:

```
shib\webAppConfig\dist.idp-container-security-example.xml
```

Debemos añadir dos elementos a dicho archivo. El primero de ellos aparece a continuación:

```
<security-constraint>
  <web-resource-collection>
     <web-resource-name>Shibboleth SSO Service</web-resource-name>
     <url-pattern>/SSO</url-pattern>
  </web-resource-collection>
</security-constraint>
```

Figura 33: Configuración de subruta /SSO

Esto le indica a Tomcat que aplique seguridad a la subruta /SSO dentro de la ruta general de la aplicación web shibboleth-idp.

El otro elemento tiene este aspecto:

```
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>UserDatabaseRealm</realm-name>
</login-config>
```

Figura 34: Configuración del archivo de contraseñas

Con esto logramos que Tomcat, en cada llamada a la ruta /SSO, pida un nombre de usuario y una contraseña, que se compararán con lo almacenado en el archivo tomcat-users.xml.

El fragmento completo que debe incluir el descriptor de despliegue de shibboleth-idp se muestra en la figura 35.

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Shibboleth SSO Service</web-resource-name>
    <url-pattern>/SSO</url-pattern>
  </web-resource-collection>
</security-constraint>

<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>UserDatabaseRealm</realm-name>
</login-config>
```

Figura 35: Configuración de la Autenticación Básica de Tomcat

Al igual que antes, ignoraremos las posibles líneas de este archivo que estén relacionadas con los roles de los usuarios.

8.5 Configurar servlets y filtros

En este apartado se mostrará cómo pueden configurarse servlets y filtros Java. Se tomará como ejemplo la aplicación web "secure". Todos los fragmentos XML que aparecen corresponden al descriptor de despliegue de dicha aplicación, que está situado en

```
\shib-filter\test-secure-application\config\web.xml
```

Incluir un servlet o filtro dentro de una aplicación web

Dentro del descriptor de despliegue de una aplicación web podemos incluir servlets y filtros Java. Para configurar un filtro, podemos utilizar las líneas que se muestran en la figura 36:

```
<filter>
  <filter-name>ShibFilter</filter-name>
  <filter-class>
  edu.internet2.middleware.shibboleth.resource.AuthenticationFilter
  </filter-class>

...
</filter>
```

Figura 36: Asociar el nombre de un filtro a una clase Java

Con esas líneas estamos definiendo la existencia de un filtro y especificando qué clase Java debe instanciarse cuando Tomcat lo ponga en marcha. Para el caso de un servlet, basta con cambiar "filter" por "servlet" en la figura anterior.

Establecer parámetros de inicialización

Siguiendo con el ejemplo de la aplicación "secure", podemos usar las líneas que aparecen en la figura 37 para dar valor a una variable al inicio de la ejecución del filtro.

```
<filter>
...
<init-param>
  <!-- Assertion Consumer Service -->
  <param-name>shireURL</param-name>
  <param-value>
  https://sp.example.org:9443/shibboleth-sp/Shibboleth.sso/SAML/POST
  </param-value>
</init-param>
...
</filter>
```

Figura 37: Un ejemplo de parámetro de inicialización de un filtro servlet

En concreto, estamos estableciendo el valor de la dirección del Assertion Consumer Service del Proveedor de Servicios.

Asociar un filtro con el acceso a un tipo de archivo

Para que el filtro de acceso se ponga en funcionamiento cuando se intente acceder a determinados tipos de archivos usamos la directiva de Tomcat `<filter-mapping>`. Lo vemos en la figura 38:

```
<filter>
...
  <filter-mapping>
    <filter-name>ShibFilter</filter-name>
    <url-pattern>*.htm</url-pattern>
  </filter-mapping>
...
</filter>
```

Figura 38:Asociación de un filtro a un determinado tipo de archivo

En este caso, el filtro entrará en acción si se llama a cualquier archivo dentro de la ruta asignada a la aplicación web, que tenga extensión “htm”.

Otra forma de configurar qué archivos se protegen

Acabamos de configurar el filtro de acceso para que se ejecute sólo si intentamos acceder a archivos con extensión "htm". Una vez que se ponga en marcha, el código de nuestro filtro (contenido en la clase que aparecía en la figura 36) realizará otro análisis del nombre del recurso deseado, comprobando si encaja con el patrón proporcionado por el parámetro de inicialización "requireId". La forma de establecer el valor de este parámetro es añadir las líneas que vemos en la figura 39, de nuevo al descriptor de despliegue de "secure".

```
<filter>
...
  <init-param>
    <param-name>requireId</param-name>
    <param-value>.+/historial.+</param-value>
  </init-param>
...
</filter>
```

Figura 39: Parámetro de inicialización del filtro de acceso

En este caso se configura para que se proteja el acceso a todos los archivos dentro de la ruta de la aplicación web, que incluyan en su nombre la palabra "historial".

Hay que hacer notar que el formato para especificar este parámetro es el de una "regular expression", mientras que el que aparecía en la directiva `<filter-mapping>` corresponde al de una "wildcard". Son, por tanto, dos sintaxis diferentes.

8.6 Configurar la gestión de identidad

Tras el proceso de autenticación del usuario que lleva a cabo el Proveedor de Identidad, se envía una aserción SAML al Proveedor de Servicios. Para identificar al sujeto al que hace referencia dicha aserción pueden usarse diferentes elementos. El elemento elegido será empleado también por el propio Proveedor de Servicios para referirse al usuario, si solicita posteriormente sus atributos.

A continuación vamos a analizar por separado cómo se configura el Proveedor de Identidad y el de Servicios para que utilicen un determinado elemento como identificador del usuario. Los archivos de configuración a los que hacemos referencia pueden encontrarse (a menos que modifiquemos el proceso que lleva a cabo Ant) en alguna en las ubicaciones siguientes, según se refieran a la configuración del Proveedor de Identidad o al Proveedor de Servicios:

```
C:\usr\local\shibboleth-idp\etc
C:\usr\local\shibboleth-sp\etc
```

Puede encontrarse información más precisa sobre las ubicaciones de los archivos en el apartado 8.8. Sobre gestión de identidad hablaremos de nuevo en 10.3.

Identity Provider

En idp.xml nos encontramos con dos fragmentos relativos a esta cuestión. En el primero, que aparece a continuación, establecemos un nombre simbólico ("shm") para el identificador que usaremos:

```
<NameID nameMapping="shm"/>
```

Este nombre simbólico debe corresponderse con un elemento `<NameMapping>` del mismo archivo, en el que se declare el tipo de identificador deseado. Vemos esto en el siguiente fragmento:

```
<NameMapping id="shm" type="SharedMemoryShibHandle"/>
```

Establecemos ahí que usaremos un handle, un identificador numérico. Otra opción sería `Principal`, con lo que usaríamos el propio nombre del usuario (el nombre que él mismo ha introducido en la máscara de la Autenticación Básica) para referirnos a él. En nuestro modelo, dicho nombre representaría su identidad federada.

Service Provider

En esta cuestión, el Service Provider se limita a seguir lo establecido por el Identity Provider. Ya esté para usar un handle o la identidad federada del usuario, el Service Provider usará lo mismo. No hay, por tanto, nada que configurar en él, relacionado con este tema.

8.7 Configurar la gestión de atributos

Con respecto a cómo se manejan los atributos en el sistema, hay que configurar de dónde los lee el Proveedor de Identidad (origen o fuente de atributos), qué atributos estará autorizado a revelar (Attribute Release Policy) y, finalmente, cuáles de esos atributos tendrá en cuenta el Proveedor de Servicios a la hora de tomar la decisión de acceso (Attribute Acceptance Policy).

Los archivos donde se configura todo esto están, al igual que en el apartado anterior, en las ubicaciones siguientes:

```
C:\usr\local\shibboleth-idp\etc
C:\usr\local\shibboleth-sp\etc
```

Volveremos a hablar sobre gestión de atributos en el apartado 10.4. En 8.8 puede encontrarse una tabla con la ubicación exacta de los archivos a los que nos referimos en este apartado.

A continuación, analizamos por separado la configuración relativa a atributos en el Identity Provider y en el Service Provider.

Identity Provider

- **Fuente de atributos**

  Dentro del archivo de configuración general idp.xml, el elemento `<IdPConfig>` posee (entre otras) la propiedad `resolverConfig`, que designa el nombre del fichero donde se almacena la relación entre cada atributo y su origen correspondiente, en este caso resolver.xml:

```
<IdPConfig resolverConfig =
"file:/C:/usr/local/shibboleth-idp/etc/resolver.xml">
```

  Dentro del archivo resolver.xml asociamos una clase Java a cada atributo que deseemos gestionar. A esta clase se la denomina **conector** y se ejecutará siempre que la Attribute Authority requiera obtener el valor de su atributo asociado. En primer lugar, asociamos al atributo eduPersonAffiliation un conector con nombre simbólico "echo":

```
<SimpleAttributeDefinition id =
"urn:mace:dir:attribute-def:eduPersonAffiliation">
     <DataConnectorDependency requires="echo"/>
</SimpleAttributeDefinition>
```

Figura 40: Conector para un atributo

  Seguidamente definimos la clase conector que asociamos a dicho nombre simbólico:

```
  <CustomDataConnector id="echo"
class="edu.internet2.middleware.shibboleth.aa.attrresolv.provider.
  SampleConnector"/>
```

  Esta clase está diseñada sólo para un sistema en pruebas, puesto que se limita a devolver siempre "member" como valor para el atributo pedido. Para un caso

real, sustituiríamos el archivo resolver.xml por otros como resolver.jdbc.xml o resolver.saml.xml (incluidos dentro de las carpetas de los proyectos Java), que nos sirven como ejemplo para establecer bases de datos o directorios LDAP como fuente de atributos.

- **Attribute Release Policy (ARP)**

  También en el archivo idp.xml encontramos la configuración referente a los atributos que pueden ser revelados ante una petición. Dentro del elemento `<ArpRepository>` declararemos la ubicación de los archivos donde residirá la configuración de la política de entrega de atributos (`<Path>`) y la clase que se encargará de analizar y gestionar el contenido de dichos archivos (propiedad `implementation`). Éste es el correspondiente fragmento de idp.xml:

```
<ReleasePolicyEngine>
      <ArpRepository
      implementation="edu.internet2.middleware.shibboleth.
      aa.arp.provider.FileSystemArpRepository">
  <Path>$SHIB_HOME$/etc/arps/</Path>
 </ArpRepository>
</ReleasePolicyEngine>
```

Figura 41: Configuración de la ARP

La variable `$SHIB_HOME$` será sustituida durante el proceso de compilación / construcción por C:\usr\local\shibboleth-idp.

Así pues, nos dirigimos a la ubicación

`C:\usr\local\shibboleth-idp\etc\arps`

donde reside el archivo arp.site.xml. Éste define una política de entrega de atributos que tendrá validez general para todo el Identity Provider. En nuestro caso, no tenemos políticas particulares para cada usuario, así que lo que se

defina en arp.site.xml constituirá la política efectiva. En las líneas que aparecen a continuación, pertenecientes a dicho fichero, permitimos la entrega de un atributo en concreto, a cualquier Service Provider perteneciente a la federación que lo solicite y sea cual sea el valor de dicho atributo:

```
<Target>
      <AnyTarget/>
</Target>
<Attribute
name="urn:mace:dir:attribute-def:eduPersonAffiliation">
      <AnyValue release="permit"/>
</Attribute>
```

Figura 42: Configuración de un atributo

Service Provider

Una vez que el Service Provider ha recibido del agente de credenciales (o de la Attribute Authority, en nuestro caso) los atributos del usuario, filtra aquellos que le interesan con relación a la decisión de acceso. Las reglas para realizar esta operación se encuentran establecidas en la Attribute Acceptance Policy (AAP) y residen en el archivo `AAP.xml`. Todos los atributos no nombrados en este archivo serán filtrados, no serán tenidos en cuenta. En el siguiente fragmento de AAP.xml, configuramos el sistema para recibir el valor del atributo que aparece en la propiedad `Name`, sea cual sea el servidor de procedencia de dicho atributo (`<AnySite>`):

```
<AttributeRule Name="urn:mace:dir:attribute-
def:eduPersonAffiliation" CaseSensitive="false" Header="Shib-EP-
UnscopedAffiliation" Alias="unscoped-affiliation">
      <AnySite>
            <Value>MEMBER</Value>
            <Value>FACULTY</Value>
            <Value>STUDENT</Value>
            <Value>STAFF</Value>
            <Value>ALUM</Value>
            <Value>AFFILIATE</Value>
            <Value>EMPLOYEE</Value>
      </AnySite>
</AttributeRule>
```

Figura 43: Fragmento de AAP.xml

No aceptaremos un valor para el atributo que no pertenezca al conjunto enumerado usando los elementos `<Value>`.

8.8 Hacer permanente la configuración

Como se ha comentado a lo largo de todo este capítulo, tanto el Service Provider como el Identity Provider poseen carpetas locales independientes donde almacenan sus archivos de configuración. Los archivos correspondientes al Identity Provider se encuentran normalmente en

```
C:\usr\local\shibboleth-idp
```

y los del Service Provider en la ubicación

```
C:\usr\local\shibboleth-sp
```

Esto es configurable modificando el archivo build.properties, como ya se ha visto anteriormente en el apartado 8.2.

Los archivos con la configuración por defecto están dentro de las carpetas de los proyectos Java del sistema implementado, “shib” y “shib-filter”. De ahí son copiados a las ubicaciones en C:\usr\local mediante el proceso que lleva a cabo Ant.

A continuación presentamos una tabla con los principales archivos de configuración. Se indica su ubicación original dentro de las carpetas de los proyectos Java, el directorio de destino donde los copia el proceso de compilación / construcción y una breve descripción de cada uno de ellos:

Ubicación original	Ubicación tras Ant	Descripción
shib\src\conf\dist.idp.xml	C:\usr\local\shibboleth-idp\etc\idp.xml	Configuración general del IdP, valor del parámetro providerId del IdP, ubicación local del resto de archivos de configuración
shib\src\conf\arps\arp.site.xml	C:\usr\local\shibboleth-idp\etc\arp.site.xml	Configuración de la Attribute Release Policy(ARP)
shib\src\conf\resolver.xml	C:\usr\local\shibboleth-idp\etc\resolver.xml	Configuración del origen de los atributos
shib\src\conf\dist.sp.xml	C:\usr\local\shibboleth-sp\etc\sp.xml	Configuración general del SP, valor del parámetro providerId del SP
shib\src\conf\AAP.xml	C:\usr\local\shibboleth-sp\etc\AAP.xml	Configuración de la Attribute Acceptance Policy(AAP)

Figura 44: Principales archivos de configuración del sistema

Para modificar la configuración del sistema, basta con alterar el contenido de los archivos listados en la columna “Ubicación tras Ant” y reiniciar Tomcat.

El proceso de compilación / construcción del código hace que el sistema vuelva a su configuración por defecto, borrando los archivos de la segunda columna y sustituyéndolos por los que aparecen en la primera. Son estos últimos, por tanto, los que debemos modificar para que los cambios en la configuración sean permanentes.

En un entorno real las carpetas dedicadas al IdP y al SP estarían alojadas en sus respectivos servidores y no en un mismo ordenador, como en nuestro caso.